# HARDWARE MANUAL

Contents

[HARDWARE MANUAL 1](#_Toc487554785)

[Hardware 1](#_Toc487554786)

[Setup 1](#_Toc487554787)

[Communication 2](#_Toc487554788)

[Node MCU 2](#_Toc487554789)

[Sensor module 2](#_Toc487554790)

[Actuator module 3](#_Toc487554791)

[ATtiny 3](#_Toc487554792)

[Sensor modules 3](#_Toc487554793)

[Actuator modules 4](#_Toc487554794)

[PCB 4](#_Toc487554795)

[Design 4](#_Toc487554796)

# Hardware

## Setup

You are going to need the Arduino IDE to compile and upload the software to all the devices.

Libraries

You need to install the ESP8266 library   
TinyWireS libraries (for the slave modules on the ATtiny85)  
usiTwiSlave   
Wire.h  
Openwifi.h  
WiFiManager.h

Boards

ATtiny85   
NodeMCU  
Digispark

You also need the usb driver if you’re using the digispark ATtiny during development.

In order to install software to the ATtiny85 you need another device to act as the programmer. The ATtiny85 can’t be connected to the computer directly so we used an Arduino Mini to install software on each of the ATtinys used.

Connect the Arduino to the ATtiny as follows:

* Arduino +5V ---> ATtiny Pin 8
* Arduino Ground ---> ATtiny Pin 4
* Arduino Pin 10 ---> ATtiny Pin 1
* Arduino Pin 11 ---> ATtiny Pin 5
* Arduino Pin 12 ---> ATtiny Pin 6
* Arduino Pin 13 ---> ATtiny Pin 7

Then you need to install the code for the Arduino to make it function as the programmer. Take the ArduinoISP code from the Examples and upload it to the Arduino. Now select the Attiny85 as board in the IDE and change the bootloader mode from ASVRII to Arduino as ISP.

Now you can upload the code to the ATtiny85 using the Arduino.

For more information, you can follow the full tutorial here:  
http://www.instructables.com/id/Program-an-ATtiny-with-Arduino/

## Communication

The communication between the ATtiny85 modules is done via I2C. It uses 2 pins for communication and can be daisy-chained meaning you can connect multiple devices on the same line and communicate with all of them using their address.

On the NodeMCU the “Wire.h” library is used to transmit and receive data. The ATtiny85 need the TinyWire library to communicate using I2C. This is an opensource library.

The NodeMCUs (both sensor/actuator modules) function as the master and the ATtiny85s (sensor/actuator modules) function as the slave. This means that the NodeMCUs needs to initiate communication.

# Node MCU

## Sensor module

The sensor module uses the SensorModule.ino code for the NodeMCU.

**Wi-Fi**

The config.h library contains the first part of the URL to the API (E.g. <http://iotworkshop.aqiqi.nl/>) The second part is added in the setup(). The config.h also contains the WIFI SSID/password for the backup WIFI for when you want to use your own Wi-Fi network.

In the MaterNode.ino it connects to the OpenWifi from the HVA, to connect to this Wi-Fi you need the openwifi.h library because the open Wi-Fi network requires to accept the terms and agreement before allowing you to connect to the internet.

**getData() function**

The getData function on the sensor module uses the “Wire.h” library. It starts by transmitting a check value of 0xAA (170) to the designated device(sensorId). Then it waits for 5 ms and expects a 3 bytes package from the slave device. Then it loops the number of bytes that come in and puts the values in the variables.

**Communication with the API**

The sendButtonPress function which happens every half of a second and calls the HTTPClient function with the second part of the url that goes to the API and includes the chipID. (Wordt dit nog gebruikt?)

De requestMessage functie has 2 requestStrings one for requesting the sensor ID because the device doesn’t initially know which slave device is connected to it. If sensorId is 0 it uses requestString2. If the sensorId is known it uses requestString to send the values that it got from the getData function.

The http.GET() does a get request to the server, and if it is successful it will echo from the API that the insert function has been successful.

The API looks for an url with the format (?deviceId=<id>&deviceFunctie=sensor&sensorId=<sensorID>&value=<sensorValue>)

Or

(?deviceId=<id>&deviceFunctie=sensor&sensorId=<sensorID>)

The deviceFunction value indicates if the device is a sensor module or actuator module for the API.

Based on the device ID the API knows which sensor is connected to the Master module because it is specified by the user on the website.

## Actuator module

The actuator module is very similar to the sensor module but it has the sendData function instead and doesn’t send a value to the database but instead receives the value of the Sensor Module and sends that to the actuator slave device (a LED for example). After it receives the value from the API it will call the SendData function with the value as parameter.

The SendData function first prints the value it is going to send to the slave module and then starts transmitting it to the actuator device with the ID it received from the API.   
After 5ms it checks the value that the slave module has received and sent back together with the address of the slave module to see if there hasn’t gone anything wrong during transmission and to help bug fixing.

# ATtiny

The Digispark ATtiny85 were used during development. Installing software on the ATtiny chip is possible by using an Arduino Nano/Mini/Uno as ISP programmer. The slave devices have an example for sensor and actuator modules.

The sensor modules have an I2C address hardcoded, each module (actuator & sensor) must have a unique ID. We have reserved 001 to 199 for the sensor modules and 201 to 255 for the actuator modules. The I2C address and the sensor/actuator IDS are equal to each other meaning that if you add another sensor module you must hardcode the same ID that it has on the database in the ATtiny code.

The code written for the ATtiny modules include code to communicate with the master and code to read/write from/to the sensor/actuator.

## Sensor modules

The sensor module starts with defining the address and the pins that are needed for the sensor. The “lichtWaarde” variable is an unsigned integer that will contain the value of the senor and the “byteRcvd” variable contains the check value that it will receive from the master.

In the setup() it will go into I2C slave mode where it listens to incoming communication from the master. You will also need to setup the pinMode for the sensor. (most likely to input)

In the void loop, you write the code that reads out the sensor with either a digitalRead or analogRead function. If it receives values from the master it will send the sensor value, the received check value, and his own address back to the master.

## Actuator modules

The actuator module is very similar to the slave module. It starts with defining the I2C address and the pin of the actuator. It initialises the I2C slave mode and puts the pinMode on Output.

In the void loop, it waits for an incoming connection from the master and stores the first byte it receives. Then it will send back the received byte for debugging purposes and its own address. After that the code uses that value to either power on the light if its lower than a threshold or turn of the light when there is a low amount of light measured at the sensorModule.

# PCB

The PCBS are designed using Eagle. The connections between the sensor/actuator, the ATtiny, and the 2 4-pin connectors are defined in the “.sch” file. The traces that go from pin to pin are designed in the “.brd” file.

We used the ATtiny85 library in Eagle for the schematic and board.

## Design

The ATtiny85 is placed in a way where the VCC, SCL(I2C), SDA(I2C) and pin 1 are all pointing towards the sensor, this makes It easier to connect all the traces without having to overlap them. The only downside is that Pin 1 isn’t an analogue pin so it can’t be used for sensors that need an analogue input/output. In these cases, Pin 2 or 3 are used and space is made in order for the traces to reach the sensor without interfering with the other traces.