Analog Temperature sensor tutorial

This guide is intended to illustrate how to connect and utilize Microchip's MCP9700T analog temperature sensor with LoRaWAN, The Things Network (TTN) and Ubidots dashboard. This project is easy to setup and will take an evening to complete. Created by Joseph Strandnes (JS225PY).

Objective

The objective with the project was to familiarize with LoRa, a large portion of the project was setting up LoRa and waiting for the LoRa Gateway to arrive, there is no coverage in my area. The reason I wanted to setup a temperature sensor was because I wanted to monitor the temperature in a nearby shack. The shack is out of range of WIFI and I use the shack too store homebrewed beer during the fermentation process.

Materials

Required

1. LOPY4 with LoRa antenna
2. Expansion board 3
3. MCP9700T temperature sensor
4. Connection cables

Optional

1. LoRa gateway
2. 3D-printed case
3. Battery with compatible connector

Components description

1: LOPY4 is a development board based on the ESP32 IoT SoC. The processor has a RF-core for WIFI and Bluetooth, two system cores (dual core) a Ultra Low Power (ULP) co-processor with access to the ADC, memory and other peripherals. LOPY4 also includes "external" RF circuits for LoRa and Sigfox. The components are all placed on a small dev board. The cost for the LOPY4 + antenna is ~44€, which were purchase at pycom.io.

2: The Expansion board allows the user to program the LOPY4 and other PYCOM board. It also functions as a break out board for the processor modules. The cost for the expansion board is 16€, which was purchase at pycom.io.

3: The MCP9700T is a analog temperature senor with built in linearization making it easy to work with. The out of the box accuracy is +-2 degrees Celsius but can easily be compensated down to +-0.5 degrees Celsius. The cost of the sensor is ~1.7 kr, I owned it prior to the project but it can be purchased at DigiKey, Mouser, Elfa and all other online resellers.

4: The cables are meant to connect the external sensors to the dev board. They cost a few kronor which can be purchased at most hobby stores.

Optional components

1: LoRa gateway functions as a router for LoRa and connects sensors via LoRa to the internet. The gateway I purchased is the Dragino LPS8 and costs ~900kr (Elfa). The gateway is a necessity if one lacks coverage from the LoRa network/TTN.

2: 3D-Printed case, the case can be found in this project directory and was designed in SolidWorks 2019. The case makes it easy to organize all the cables and gives the finished project a sleek design. The cost for the case is a few kronor.

3: The battery allows you to operate the LOPY4 remotely without any external power. The battery can be purchased at most hobby shops for <100kronor.

Computer setup

This section will describe the process of programming the LOPY4 device.

Required programs

- Pycom Flash Tool Firmware Updates.
- Atom IDE.
- Pymkr plugin for Atom. Can also be downloaded from the Atom package installer.
- The Things Network account with Keys

1: Use the flash tool update to the latest firmware for the LOP4. This is done by running the program Firmware upgrader and following the simple instructions in the program. Make sure to select LoRa region as "EU868".

2: Install Atom, press "ctrl+," (press control key and comma key simultaneously) and press the "package" button on the left side of the screen. Type Pymkr in the search bar, wait for the Pymkr plugin to show up, make sure that it is released by Pycom (says creator name next to the logo). Press install and you should be all set!

3: Download the project files from this GitHub repository by either pressing download on the GitHub page or by using the "Clone" command.

4: Open Atom and press "ctrl+shift+A" (Add project folder) and navigate to the folder were you downloaded this project. Select the folder "Software"

6: The File config.json contains the necessary keys for LoRa communication, add your keys from The Things Network here.

7: Press "ctrl+alt+S" (Upload project to device) to program the device.

8: It should now be connected to The Things Network and sending the measured temperature through LoRa too the TTN console.

Electrical connections

This section describes the electrical connections necessary to complete this project.

The Temperature sensor has three connections VCC, GND and OUPUT.

Temperature sensor connections

- VCC connects to 3V3 or 5V DC
- GND connects to ground

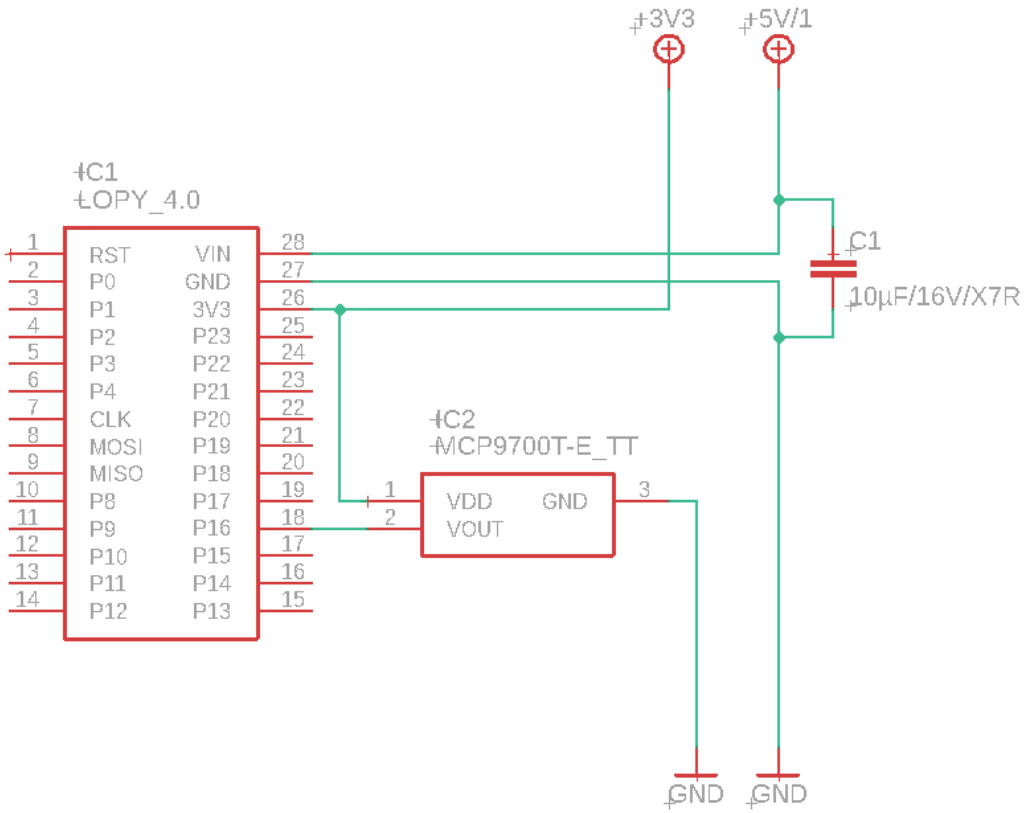
- OUTPUT should be connected to an analog pin of the LOPY4 device. Pin "P16" is used in this case

  LOPY4 connections

LOPY4 connections

- Connect VIN to 5V
- Connect GND to ground (0V)

Below is the electrical connection. The schematic was drawn in Autodesk Eagle and can be found under "Electrical" in the root directory. The temperature sensor has internal current limiting so no external components are necessary. A capacitor for the LOPY4 VCC is generally good practice to prevent unstable power rails during transmission. The recommended value is 10µF, 16V and preferably X7R temperature coefficient.



Platform

This section will describe the platform used to handle the data from the sensor/LOPY4 device. This code uses LoRaWAN over The Things Network combined with Ubidots to display the data on a dashboard. This solution is entirely based on cloud application so no local installation/configuration is needed! The Things Network simplifies LoRaWAN extremely since it handles all the server related things and there are built in configuration in my LoRa Gateway. Ubidots has a lot of different display options and is perfect for this type of project.

1. The Things Network: Payload Format
2. The Things Network: Integration
3. Ubidots: Dashboard setup

1: The Things Network: Payload Format

LoRa physical is limited to byte size packets of data, the ADC measurements of the ESP32 are 12-bit witch results in an actual memory allocation of 2 bytes (16-bit). The LOPY4 "splits" the data into two bytes and send them one after another. The "Payload Formatter" allows us to format the incoming data, in this case it is stitching data back into one 16-bit variable. Below is the code that is used to stitch the two bytes together and output it as one variable called "Temperature", the value is divided by 10 to compensate for the times 10 multiplication in the LOPY4.This is done to get 1 decimal of accuracy.

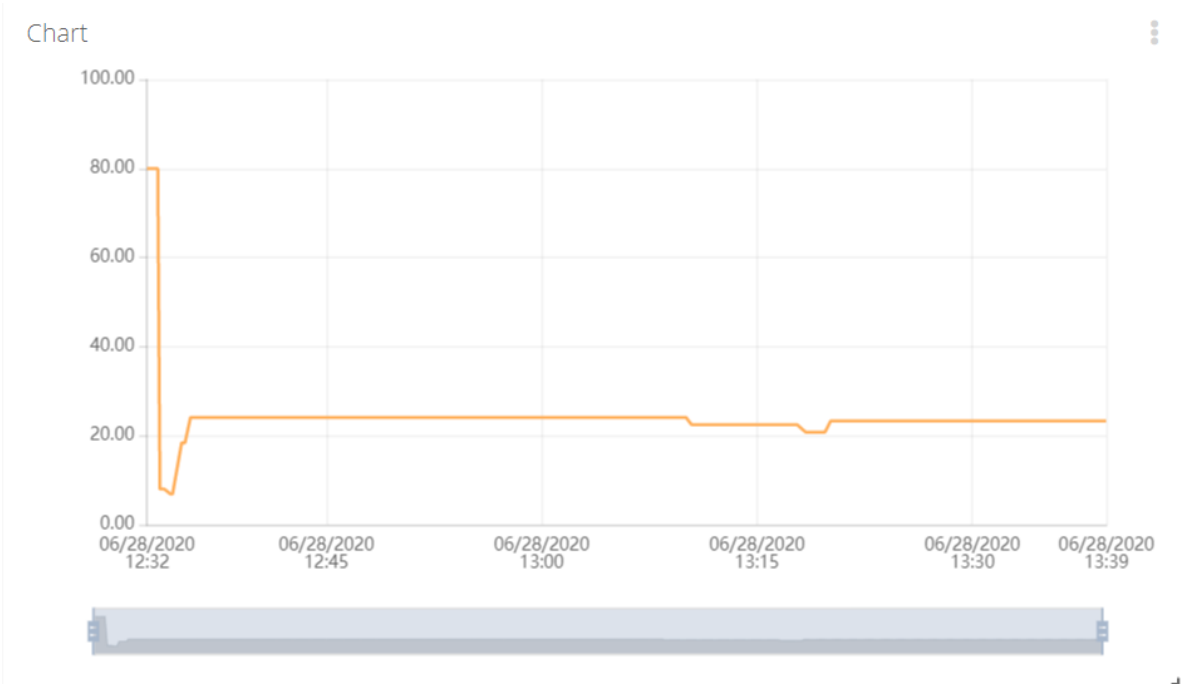
```
function Decoder(bytes, port) {

  var Temp = (bytes[0] | bytes[1])

  return {
    Temperature: Temp/10,
  }
}
```

2: The Things Network: Integration

Create an account on the Ubidots platform and copy the token from "API Credentials". Head over to The Things Network and go to the page "Integrations", press the button "add integrations" on the right side of the page and select "Ubidots". Paste the Ubidots token in the field "Ubidots Token".

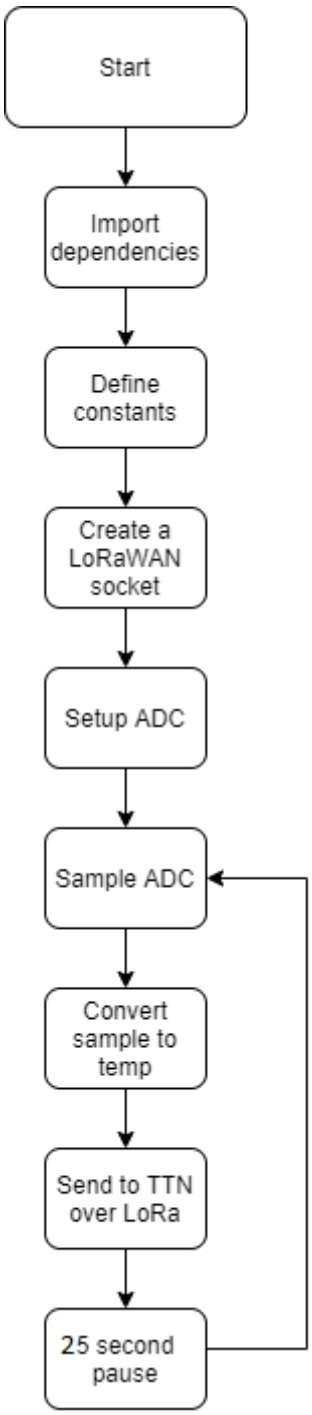
3: Ubidots: Dashboard setup

Click on the dashboard dropdown and select "Add new widget", located in the top right corner. Select the Line chart option. Press the "add variable" button and a variable named "temperature" should appear, select it. Scroll down to the Y-axis options, write "Degrees Celsius" as the Y-Axis name and add 100 as the max value on the Y-Axis, min value can remain on auto. Press the green checkmark and the line chart should appear. It should look like the image below.



The Code

The code uses the Pycom LoRa API to handle the LoRaWAN integration. The integration is handled in the standalone files lora.py and config.json, config.json is only used to store the authentication keys for The Things Network. All program code is located in the Software folder under the project root folder. The main code is located in main.py (also included below), the program is described in the flow chart below.




```
#main.py

import lora
import struct
import time
import machine
from machine import ADC

#The sensor used in this project is MCP9700T-E
#It is an cheap temperature sesnor with built in compensation and linearization
#You can compensate for self heating of teh sensor but it is not used in this
code.
#Accurecy is +-2°C
#Volatge offset at 0°C is 500mV
#Temperature coefficient is 10mV/°C
```

```
#Function to calculate temperature: Vout = Tc * Ta * V0
#Vout   = Volate output from sensor
#Tc     = Temperature Coefficient
#Ta     = Ambient temperature
#T0     = Sensor output voltage at 0°C

#Values for the temperature calculations
T0 = 500
Tc = 10
Temperature = 0

#Create a LoRa socket
lora.connect_lora()
from lora import s

#Initiate the ADC for pin 16
adc = machine.ADC()
apin = adc.channel(pin='P16')


while True:
    #Take a analog measurement
    val = apin()

    #Test Value
    #val = 900

    #Calculations for the temperature sensor
    Temperature = (val*3300)/4096               # Convert measurment to mV
    Temperature -= T0                           # Compensate for voltage offset
at 0 degrees celcuis
    Temperature /= Tc                           # Convert to temperature
measuremnt with sensor coefficient.

    #Multiply the value by 10 to get a decimal accuracy of 0.1
    Temperature *= 10                           # remember to devide by 10 on
reciveing end

    #Split the temperature measurement into two bytes (Upper and Lower)
    Lower = int(Temperature) & 0xff             # Lower byte
    Upper = int(Temperature) >> 8               # Upper byte

    #Send the two bytes to the TTN server
    s.send(bytes([Upper, Lower]))
    #print(Temperature/10)
    time.sleep(25)
```

Transmitting the data / connectivity

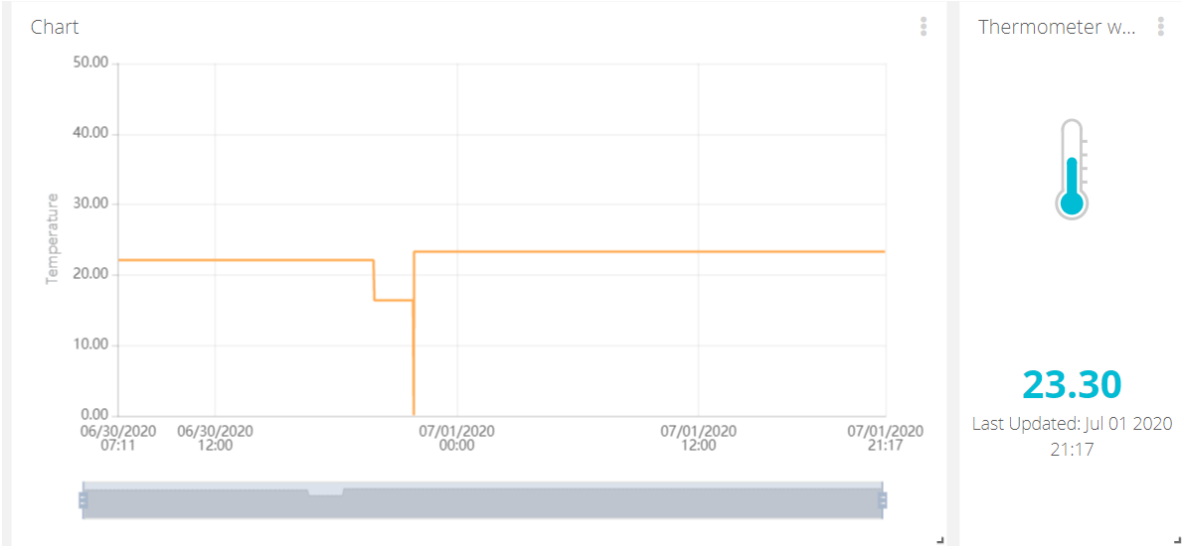
The data is transmitted over LoRaWAN to The Things Network. The data is transmitted every 25 seconds in order to minimize the amount of data and reduce the overall power draw of the system. The data rate was configured to the slowest bandwidth option (125kHz) to increase the transmission distance. The measurement data was split into two bytes since LoRa only support byte size packages/payloads, the ADC samples 12-bits of data that is stored in two bytes so the upper half of the 16-bit value was stored in a byte and the lower half is stored in another byte. Each bytes is transmitted separately, the upper byte is transmitted first, shortly followed by the lower byte. The data (payload) is stitched back together at the server by storing the upper byte in a 16-bit variable and then "oring" the lower byte into the 16-bit variable, thus returning the measurement data to its original form. Below is the function to combine the two 8-bit values to one 16-bit value.

```
var Temp = (bytes[0] | bytes[1])
```

Presenting the data

The data is "funnelled" through Ubidots API integration on The Things Networks platform. The data is also stored for seven days at The Things Network and is accessed through a REST API by using HTTP GET request, the data is fetched by my website and displayed on a graph as temperature over time (This is a work in progress and not yet finished).

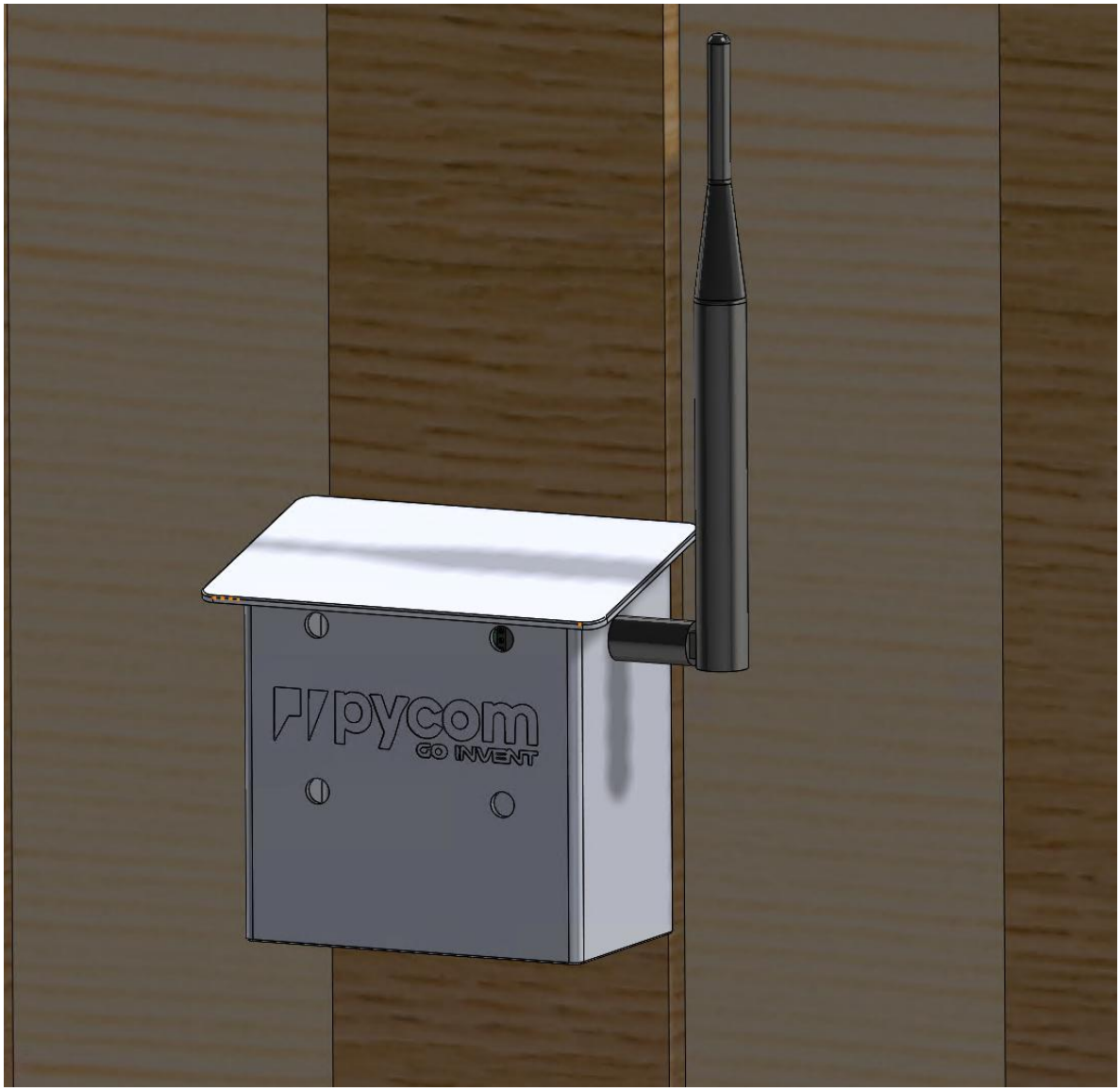
The data is displayed as single temperature reading and as a temperature over time graph on the Ubidots dashboard. The images below is during testing witch is the reason for static measurements and abrupt change of temperature.



Finalizing the design

The image below shows the final housing for the LoRa temperature transmitter. The housing was designed in SolidWorks 2019, printed on an FlashForge finder (FDM 3D-printer) and was printed from the plastic type PET. The design consists of three parts, the main housing, roof and a wall mount that attaches to the backside of the housing. It was designed to resemble a small birdhouse and have a lot of "breathing" space, the bottom is a open honeycomb mesh to allow the same temperature on the inside as the outside ambient temperature. It was not designed to be waterproof/resistant and should be mounted indoors only. The device is able to transmit data over long distances and I'm able to view the data in a convenient way.

The battery life is acceptable reaching over 24 hours in active mode, the life could be extended significantly by putting the main processor to sleep and using the ULP (Ultra Low co Processor) to measure many ADC samples, convert them to the temperature and then bulk send the data using the main processor once an hour. The processor would be put into "hibernation mode" between each measurement only drawing ~5µA and using the RTC to wake up every time it would take a sample. The battery life was not a priority due to constant access to power, the onboard ALWAYS ON LED also draws 1000 times more power than the processor would during hibernation so no additional design considerations were take in regards to power consumption.


pycom
GO INVENT