# Introduction

**Create your own plant-monitoring-system using Raspberry Pi, IEEE 802.15.4 low power wireless communication and Atmel wireless MCUs**

This article is intended for all plant enthusiasts among us, who once in a while are a little neglectfully watering them. I will show you how to set up a wireless sensor network using the IEEE 802.15.4 standard and program the Raspberry Pi to work as a network coordinator/moisture web-monitoring- system which will help you taking care of your plants.

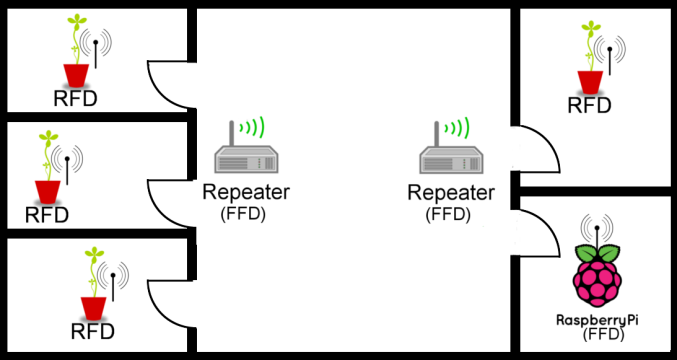
# The sensor network

The idea was to stick a battery-powered sensor in each plant pot, of which I want to measure the moisture and send the recent sensor value wireless to my Pi so it can be stored and further processed. The IEEE 802.15.4 standard defines a trans-mission protocol for Wireless Personal Area Networks. It was designed for devices with low power consumption and applications not needing high data rates. So it is just the right solution for my small sensor network.

# The network topology

Using the IEEE 802.15.4 standard entails some peculiarities. The protocol just defines the two lowest layers of the OSI Model (PHY and MAC) which means I have to take care of the network topology and manner of data sending (routing) by myself. I chose a simple tree topology, which only enables data sending in one direction, from the sensors to my Pi. The moisture-sensors (end devices) are reduced functions devices (RFDs), which are most of the time in standby mode and just wake up to measure and transmit data to the Pi. They connect to the next full function device (FFD) within range (that can be a repeater or the Pi) and are only able to send but not to receive. The Pi acts as coordinator of the network and becomes thus a FFD. Its task is to open up a network with an unique PAN id on a pre-defined network channel. If new end-devices are detected, a so called “short id” is given to them, what is a bit similar to assigning IP addresses via DHCP.

Whereas the transmission range of the RFDs is limited, I had to create some kind of repeater, which works on the base of a simple flooding algorithm. Each time incoming sensor data is received it is broadcasted to all adjacent nodes. This could be either other repeaters or the coordinator, if it is within range. To avoid duplicate deliveries to the coordinator and to prevent packets circulating infinitely in the network, every data packet gets consecutively a number, which is stored within a buffer in each FFD. If a recently received number is already in this buffer the data package is automatically discarded.

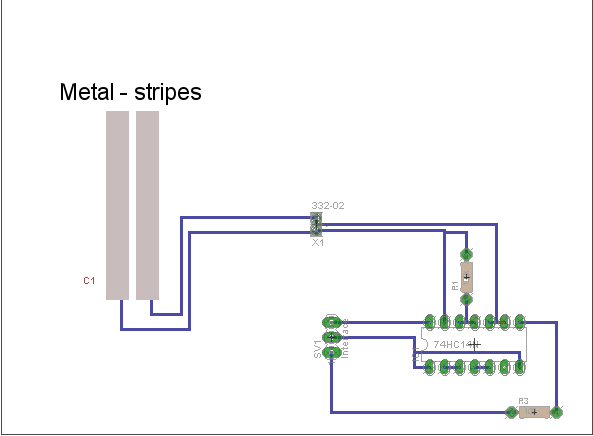


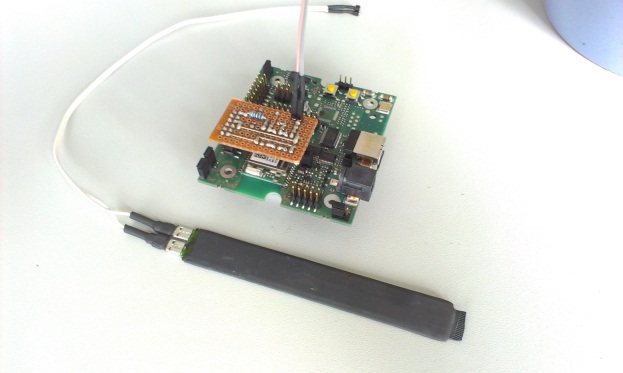
# Hardware

In order to realize the transmitters and receivers I’ve chosen Atmels RF SoCs (ATmega128RFA1/ ATmega256RFR2). They combine an 8-bit microcontroller and a RF transceiver working in the 2.4 GHz ISM frequency band having enough transmission power for our application. Since they contain an AVR core, everyone who has dealt once with AVRs should quickly get into programming the devices as well. There are manufactures, which produce complete radio modules containing not only the MCU and but also an antenna. These are much easier to handle than an IC with QFN-package! I recommend the “deRFmega128”-series of the company “dresden elektronik”. I also used their evaluation boards to develop and test my firmware.

# The moisture sensor

The moisture sensor is a relative simple DIY. It works on the principle of capacitive measurement.



Two metal strips, which are plugged into the soil creating a capacity that is, combined with a Schmitt Trigger, a RC-oscillator circuit. In order to determine the moisture level you just have to get the frequency of the circuit. Water has a much higher dielectric constant (>80) than soil (~3.9). So in wet soil the capacity of the metal stripe capacitor rises noticeable which leads to a reduction of frequency of the RC-oscillator. Enclosed by a shrink tube, the sensor might look like this:

(Sensor plus pluggable oscillator circuit for the evaluation board)

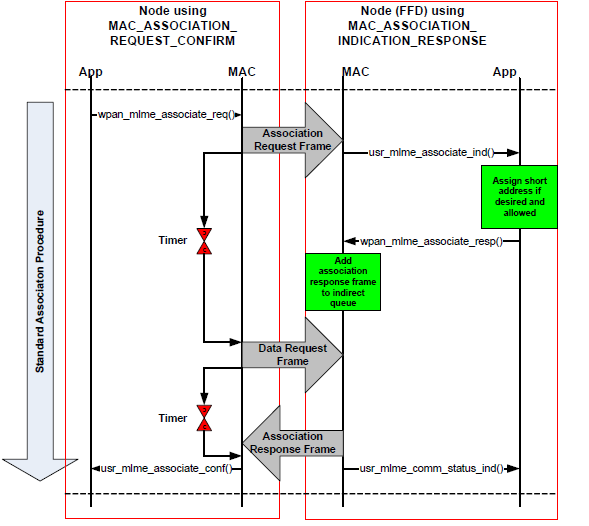
# Programming

The software for the sensors consists of 2 parts: one for measuring the frequency and analyzing the sensor values, the other part to transfer data to the Pi. Measuring the frequency is not a big deal. The circuit delivers a clock signal of about 120 kHz (depending of the size of the metal stripes) which can be easily count by an internal counter of the ATmega.

To use the integrated transceiver for sending data based on the 802.15.4 standard I downloaded the Atmel MAC protocol stack. The software is a little complicated and very extensive, so you’ll take you some time to get into it.

I just want to give a short outline:

The stack is built up of 3 layers. The Mac Core Layer (MCL), Transceiver Abstraction Layer (TAL) and the Platform Abstraction Layer (PAL). Each of these communicates with the above or underlying. The proceeding of the program takes places using a queue, on that we put data we want to transmit or from that we get error or confirm messages. Interaction with the queue usually works as follows: At first you have to make a request (e.g. to send data), which puts the instructions on the queue. After this the request has been performed, a callback function is invoked. With these confirmations you can react on errors or indicate successful transmissions.



(Standard sequence for RFD association)

Explaining more details would go beyond the scope of this article. Refer to Atmels User Guide for further information: http://www.atmel.com/Images/doc8412.pdf

With these two parts it is now possible to write a small program, which lets wake up the MCU every 6h, measure the frequency, do some sensor data validation and go to suspend mode again.

# Configuring the Pi

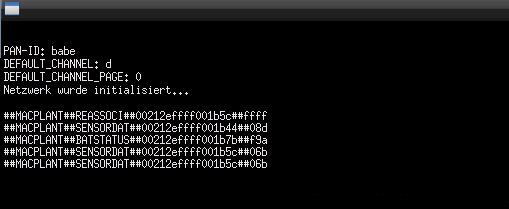
To provide the Pi access to the low power wireless world first of all we need another radio module that we can connect to it. My choice fell on the “RaspBee” also from dresden elektronik. Already designed as pluggable module it can communicate via UART with the Pi. To handle incoming data sequences I wrote a small Python script that runs permanently in background reading from UART and store data into my MySQL database. Therefore we have to install following packages:

|  |
| --- |
| sudo apt-get install python-serial python-mysqldb |

The UART can be simply accessed with the following code:

|  |
| --- |
| import serial  baudrate = 38400  byte = 8  parity = 'N'  stopbit = 1  timeout = None  xonoff = False  rts = False  dsr = False    ser = serial.Serial(‘/dev/ttyAMA0’, baudrate, byte ,parity,  stopbit ,timeout,timeout,xonoff,rts, self.dsr)  def getSensorString():  string = ""  char = ""  while (char != "\n"):  char = ser.read()  if char == '\n':  break;  string += char  return string |

My output of sensor data on the console looks like this:



These are the strings transmitted from the sensors. They contain several information: For one, the identifier ‘MACPLANT‘. This ensures our script to know that the package comes from a radio module. Afterwards follows the type of package. That is important, because the sensors not only send moistures values, but also every 3 measures the battery voltage level. Furthermore we need a type for handling devices connecting to the pi, which aren’t stored in the database yet. After that comes the MAC-address of the end device. This information has to be in the payload, because otherwise it gets lost by sending data by the repeater. At the end of the sequence we find the recently measured hexadecimal voltage (in mV) or moistures value (frequency in kHz). The line can now be parsed by our python script and stored in our database.

# Setting up webserver and database

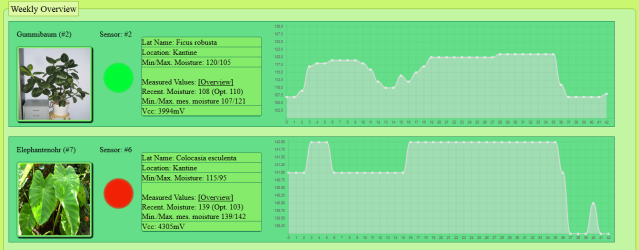
I would like to provide the measurement data in my local network. For this reason I installed a small LAMP software bundle.

|  |
| --- |
| sudo apt-get install apache2 mysql-server php5 phpmyadmin |

I used PhpMyAdmin for managing my database. If you prefer working with the console you can also use the MySQL Client software. A ready-to-use database layout is also included in the downloadable project directory.

# The web interface

Now there is just one thing left: a user interface. Based on HTML, PHP, jQuery, MySQL and the Chart.js library it is possible to create a functional, comfortable and neat GUI.



It performs the following tasks:

## 1. Show the last activities

It has a kind of logbook of the last activities, so we can see the last time sensor data was received. Additionally the system recognizes new sensors, which try to connect to the coordinator, critical battery-levels and low activity times.

## 2. Management of the plant database with the possibility to calibrate new sensors.

New plants can be added and assigned to a specific room. You can adjust the min and max moisture level of the soil, which has to be determined empiric before.

## 3. Warning at dangerously high or low moisture values.

Plants which are too dry or wet can easily be recognized the by a red or blue point in the overview. If that’s not enough you could think about other possibilities. A simple way is to install a little E-Mail server, which sends an alert to your postbox. You could also configure a Pi to SMS Gateway which sends a short text to your cellphone. Even a little sound system plugged on the Pi that plays warning messages/sounds is imaginable.

# Bottom line

The system has taken a little burden from me. Since I now will be warned of critical soil moistures, lifeless remains in my plant pots already became a rarity. If you have questions, are looking for detailed information or project files just visit the project website:

<http://rapi-projects.de.vu/>