Guide for measurements of Mobile Communications and Quantum Technologies Laboratory

Arduino/Moteino development

Place of measurement:

Department of Networked Systems and Services,
Mobile Communications and Quantum Technologies Laboratory (MCL)
I.B.113.

Made by:
Ádám Knapp

Last modification:
10. February 2017

1 Introduction

The students will learn about the concept of Internet of Things (IoT) and meet the Moteino sensor node during the measurement. It is an Arduino compatible node enabling us to build simple sensor networks. The students will learn how to program the devices, read sensor data and communicate via the radio interface of the nodes.

The Internet of Thing concept became one of the biggest buzzword in the ICT sector. It connects to such themes, like Smart Home, Smart City, Machine to Machine (M2M) communication, and of course to the mobile, wireless and sensor networks. The idea of IoT is to connect everything in a huge network. Such "thing" can be almost anything, wearable devices, household appliances, metering devices, vehicles etc. This is possible due to the different wired and wireless communication technologies. However, the IPv6 network protocol have to be highlighted, because it has advanced abilities to handle such big network e.g. its very large address domain, different discovery and security features. Therefore the IPv6 can behave as a common platform for IoT devices.

The motivation behind the IoT is making our life much easier, comfortable and much safer by automating numerous activities. This process usually starts with monitoring our environment using different sensors. Then, the generated data is aggregated, processed or sent depending on the actual implementation of the system. Finally, these information are used by the system to intervene automatically and also to provide manual control. The data processing can be done locally or remotely e.g. on a server or in cloud, and the control can be from locally or from remotely, too.

2 Preparing for the measurements

- Read this document! It contains the most important information about the measurement.
- Refresh your C programming skills! The Arduino programming reference is available in [1]!
- Think about the questions (7 Questions)!

3 Measurement layout

The students perform the tasks independently. The devices for the measurement is the followings:

- a PC with Arduino IDE,
- a Moteino sensor node [2],
- a USB-UART converter.

3.1 Moteino sensor node – RFM69 RF chip

The Moteino node is a low-price, low-power, open-source Arduino compatible platform. Its panel has an ATmega328 microprocessor as its "brain", an RFM69 RF chip and many pins to connect different sensors and other peripheral units. The microcontroller communicates with the PC via serial port, but it needs a UART-USB serial converter for proper connection. The converter also provides the 5V power for the node. The serial interface is used for uploading the code and for sending messages e.g. for debugging. Figure 1 shows the Moteino sensor node from below and above. Figure 2 illustrates the implemented packet structures by the RFM69 library. More information can be found about the node in [2], about the RFM69 RF chip in [3] and [4].

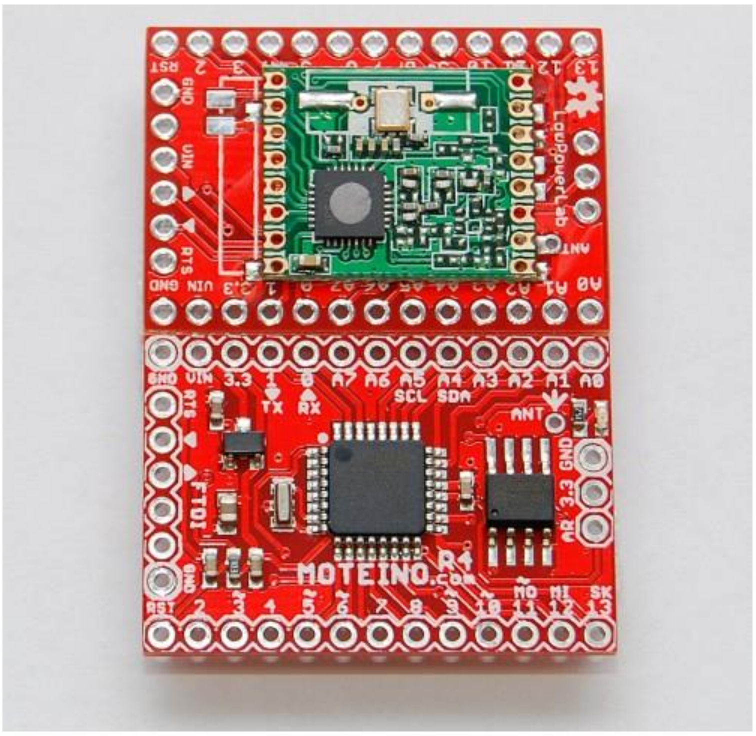


Figure 1. Moteino sensor node [2]

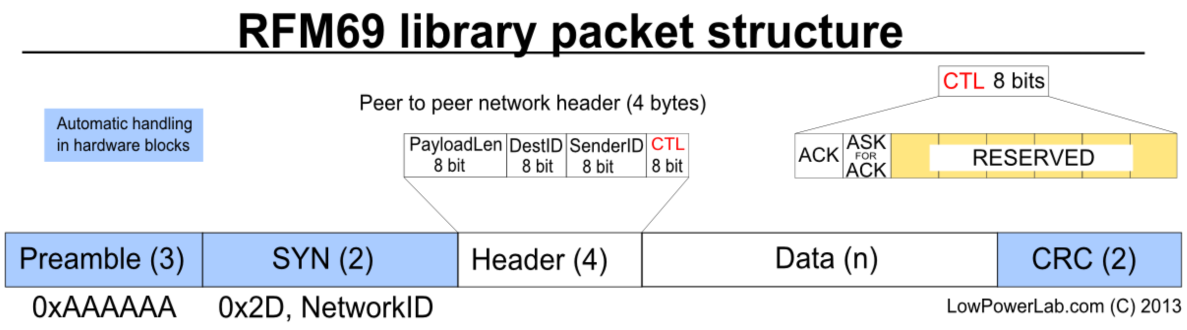


Figure 2. RFM69 packet structure [4]

3.2 Arduino IDE

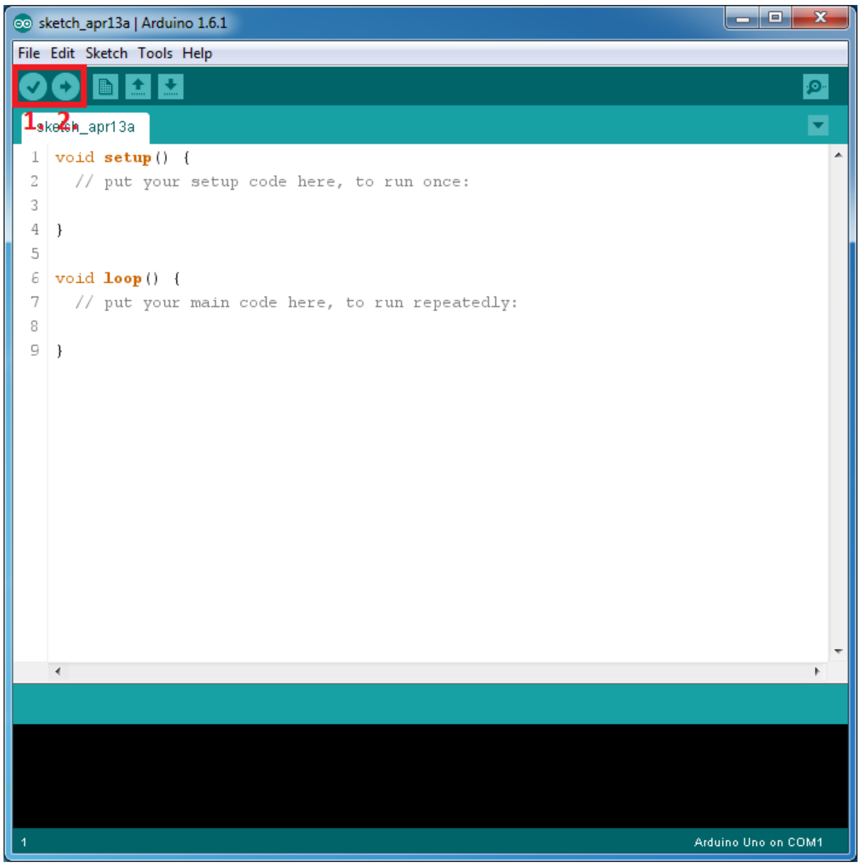
The Arduino IDE is an open-source development environment made for programming Arduino compatible devices. It has two main functions: it supports coding on a C-like Arduino specific language and it is able to compile the source code and upload that on the device.

Before the code is compiled, check the following settings (once enough after running the IDE):

- *Tools/Board* - *Arduino Uno* have to be selected,
- *Tools/Port* - select *COMX*, which represents the connected UART-USB converter,
- *Tools/Programmer* - *AVRISP mkII* have to be selected.

Follow these instructions for uploading the code (Figure 3):

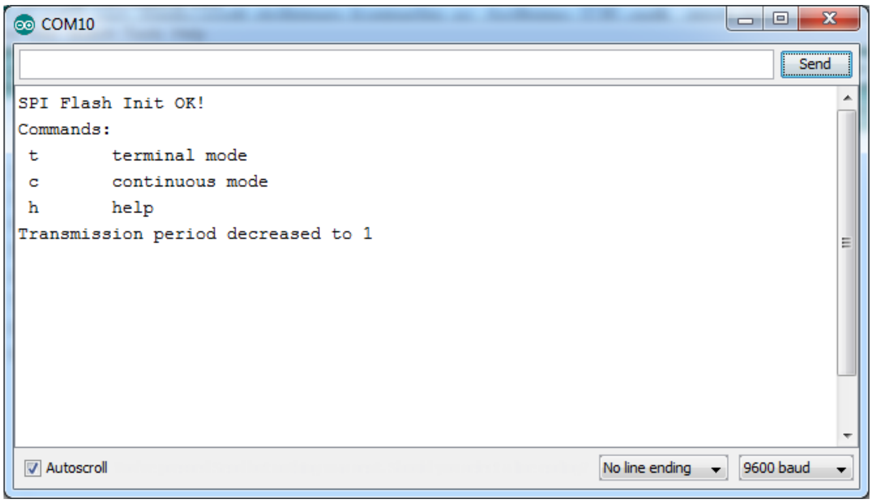
1. Verify the source code with Arduino IDE! Click on the *Verify* icon or use *Ctrl+R*!
2. Correct both errors and warnings too! If the compiling was successful, upload the code using the *Upload* icon or *Crtl+U*!



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Figure 3. Arduino IDE

Finally, one other function of Arduino IDE have to be highlighted. The *Serial Monitor* provides connection via the serial interface. It can send messages and display the received ones to/from the Moteino sensor node. It can be started in a separate window by clicking *Tools/Serial Monitor* in the menu or using *Crtl+Shift+M*. Pay attention to select the proper baudrate from the select list in the right bottom corner (Figure 4)!



```
SPI Flash Init OK!
Commands:
 t       terminal mode
 c       continuous mode
 h       help
Transmission period decreased to 1
```

Figure 4. Serial Monitor

4 Programming Arduino devices

Arduino devices work as embedded systems, therefore they have some specialties, which have to be taken into account. One of them is the limited computational and memory capacity. However, nowadays compilers are able to handle higher programming languages, than the machine code. Hence, we can write source code in C-like language developed for Arduino [1]. Nevertheless, we have to pay attention to use only the necessary number of variables and compress the source code. The most important libraries and functions are presented in the followings, which are used during the measurement. More information can be found about the Arduino specific C language in [1] and about the FRM69 library in [4].

4.1 Structure of Arduino applications

The Arduino source code can be separated logically into three main parts. The first section contains instructions for the preprocessor. This section is optional, but we have to usually load libraries and APIs for the given, specific device (in our case we load the RFM69 library). The second part contains the initialization instructions. These are executed after powering up the device, and they have to bring the device in a well-defined state. The initialization must be written in `void setup()` procedure. Finally, the third section contains the periodically executed instructions (since the embedded systems execute the same program continuously and endlessly). These instructions must be written in `void loop()` procedure. Both the initialization and the loop code parts are mandatory. Naturally, we can write other functions, procedures, libraries etc. for easier code reading and for code re-usage like in traditional C language.

4.2 Instructions for preprocessor

Two general and very often used instructions belongs here:

`#include <file>`	Loads libraries and APIs
`#define constantName value`	Defines constant

4.3 Communication

Moteino nodes have two interfaces for communication. They are equipped RFM69 RF chips for wireless communication, and USB-UART serial converter is used for serial communication with PC. The serial interface is used for programming the device, debugging the code, to issue instructions and to display data. The following table contains the most important functions for serial communication:

`byte Serial.available()`	Gets the number of bytes, which are ready to read from the serial interface
`void Serial.begin(baudrate)`	Initializes the serial interface with given baudrate
`byte Serial.print(val,format)`	Prints text or a variable via the serial interface
`byte Serial.println(val,format)`	Same as above, just prints a carriage return character and a newline character
`byte Serial.read()`	Reads the first byte of incoming serial data
`long Serial.parseInt()`	Looks for the next valid integer in the incoming serial stream

The radio interface is used for communicating between Moteino nodes. The following table contains the most important functions for RF communication:

`RFM69 radio;`	Instantiation of an RFM69 type radio object
`bool radio.initialize( byte freqBand, byte ID, byte networkID=1);`	Initializes radio; parameters: frequency band, ID of node, ID of network
`void radio.send( byte toAddress, const void* buffer, byte bufferSize, bool requestACK=false);`	Sends a packet/message; parameters: destination ID, data, data size, request for acknowledgement (ACK)
`bool radio.sendWithRetry( byte toAddress, const void* buffer, byte bufferSize, byte retries=2, byte retryWaitTime=15);`	Sends packet/message with retry; parameters: destination ID, data, data size, max. number of retries, time interval between retries
`bool radio.ACKReceived( byte fromNodeID);`	Checks that ACK is received or not; parameter: source node ID
`void radio.promiscuous( bool onOff=true);`	Receives and processes every packets independently who is the destination; parameter: turn on/off
`bool radio.receiveDone();`	Checks that the packet from the radio interface is received completely
`byte radio.SENDERID;`	Data member of sender ID
`byte radio.DATALEN;`	Data member of the received packet length (≤ 61)
`byte radio.DATA[DATALEN];`	Data member of the received packet

4.4 Handling LED and sensors

There are a LED and a temperature sensor available on Moteino nodes. In addition, the RF chip provides Received Signal Strength Indicator (RSSI) data as the descriptor of the radio channel quality. The LED can be operated by the following instructions (the PIN number of the LED is 9):

`void delay(unsigned long ms)`	Delay; parameter: milliseconds
`void digitalWrite(byte pin, value)`	Set PIN in high or low power state; parameters: PIN number, value: `HIGH/LOW`
`void pinMode(byte pin, mode)`	Configure PIN as input/output; parameters: PIN number, mode: `INPUT/OUTPUT/INPUT_PULLUP`

The temperature sensor can be accessible by the following function:

`byte readTemperature(byte calFactor=0);`	Reading temperature; parameter: correction factor

The RSSI data can be read from the RF chip by the next function:

`int readRSSI(bool forceTrigger=false);`	Reading RSSI; parameter: force immediate reading

5 Protocol specification

One goal of the measurement is to implement a simple protocol, which is capable of transferring commands and sensor data in the network of Moteino nodes. Three different logical entities are distinguished in that sensor network (each node is configured manually):

- Slave: it reads its sensors and sends the measurement data periodically (~1-2 seconds). It also responds to the received commands.
- Aggregator: it has similar operation, than the slave, but it also aggregates sensor data from connected slaves and sends them to the master. It also forwards commands from the master to the proper slave.
- Master: it has similar operation, than the aggregator, but it displays the received and locally generated sensor data. It processes commands from the PC and sends them to the proper nodes.

The packet format of the protocol is the following:

Header 1 byte	Destination address 1 byte	Data 1. byte	Data 2. byte	Data 3. byte	Data 4. byte

Figure 5. Packet structure of the simple sensor network protocol

This packet is transmitted in the data field of the RFM69 packet (Figure 2). This packet is fixed 6 bytes long, therefore no length field is necessary (but the RFM69 packet contains such field in its header anyway). Two types of data is allowed in the data field: temperature and RSSI. The value of the header field influences the node operation as follows:

Header byte value	*Description*
1	Data field contains temperature value
2	Data field contains RSSI value
10	LED is turned on (data field set to 0)
11	LED is turned off (data field set to 0)
12	LED is blinking (data field set to 0)
20	Turn on temperature sensor (its value is read periodically)
21	Turn off temperature sensor (its value is not read)
22	Turn on RSSI sensor (its value is read periodically)
23	Turn off RSSI sensor (its value is not read)

6 Measurement tasks

Task 1.

Introduction of using Moteino sensor node: connecting to PC, coding and uploading the first application.

Task 2.

Communicating with PC using the serial port: getting and displaying data, getting commands.

Task 3.

Usage of LED and sensors of Moteino node: turning on/off and blinking the LED, reading temperature and RSSI data from sensors.

Task 4.

a) Sending message via the radio interface: RF initialization, sending and receiving packets.

b) Implementing the simple sensor network protocol: implementing the packet structure, sending and receiving packets wireless, assembling and processing packets.

Task 5.

Establishing a sensor network and testing the implemented functions.

7 Questions

- What kind of modulation (type, number of states, other attributes) do the RFM69 radio use? (It can be found in the RFM69 chip specification.)
- What kind of transmitter frequency bands are supported by the RFM69 radio? (It can be found in the RFM69 chip specification.)
- Define a structure on C programming language, which contains 1 byte header and 7 byte data field!
- Which functions can be used for getting the RSSI and temperature values?
- Which functions are necessary to be implemented for Arduino based embedded system?

References

[1] Arduino Language Reference, https://www.arduino.cc/en/Reference/HomePage

[2] All about Moteino, http://lowpowerlab.com/moteino/

[3] RFM69 ISM TRANSCEIVER MODULE V1.3, http://mcl.hu/?q=hu/laboratory/usrp

[4] RFM69 library, http://lowpowerlab.com/blog/2013/06/20/rfm69-library/