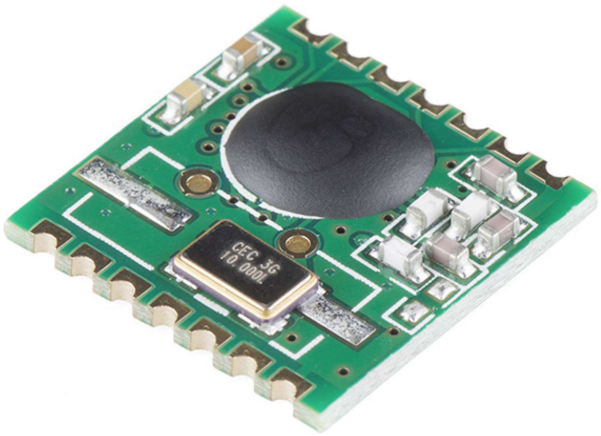
Lab 2

Broadcasting Data with the RFM12B-S2 Radio Module

The aim of this second lab is to become familiar with basic network functionality using the RFM12B wireless transceiver and Il Matto development board. This will build on the first lab (getting two boards to communicate), by exploring link access and broadcasting data with the RFM12B-S2 module across a shared channel.



Schedule

Preparation time	:	n/a
Lab time	:	3 hours

Items provided

Tools	:	n/a
Components	:	RFM12B-S2 Radio Module (1 each)
Equipment	:	Oscilloscope, Logic Analyser
Software	:	n/a

Items to bring

Essentials. A full list is available on the Laboratory website at https://secure.ecs.soton.ac.uk/notes/ellabs/databook/essentials/

Revision History

October 14, 2016	Domenico Balsamo and Geoff Merrett	Minor revisions for 2016/17
October 13, 2015	Domenico Balsamo and Geoff Merrett	New lab for 2015/16

1 Aims, Learning Outcomes and Outline

These lab exercises will provide an introduction to computer networks and wireless communication using the Il Matto development board. In particular, you will learn how to use the RFM12B-S2 wireless transceiver (transceiver = receiver + transmitter) to enable communication between two Il Matto boards, and the basics of how to implement a protocol stack.

In this lab you will explore link access and broadcasting data across a shared channel with the RFM12B-S2 module. Link access is one of different services provided by the Data Link Layer. Other services are, for example, framing, flow control, error detection and correction etc. You will implement different Carrier Sense Multiple Access (CSMA) protocols (0-persistent CSMA, and either 1-persistent or *p*-persistent CSMA). **In this second lab you should not worry about protocol layering**. We will explore this in the third and final lab.

---

2 Preparation

None required.

---

3 Laboratory work

You should work in pairs for this lab exercise. Each pair should ensure that they have at least one working Il Matto between them, and that at least one person is familiar with the Il Matto and its programming.

Your objective for this lab is to explore link access and broadcasting data across a shared channel using the RFM12B-S2 module, by implementing CSMA protocols (0-persistent CSMA and either 1-persistent and/or *p*-persistent CSMA – it is up to you).

The data being transmitted should be an 8-byte ASCII string representing your own (suitably imaginative) lab-pair 'team name', and everyone should be communicating over the same common channel (at least until you get onto the optional extension at the end of these lab notes). **You should all discuss with each other (i.e. everyone else doing this lab) to agree on the frequency/channel you are all going to use.** Whenever your system is not transmitting, it should be listening to the channel and receiving data that anyone else is sending.

Received data should be printed on the terminal via the Il Matto's serial connection. Your 'team name' can be hard-coded into your code, or entered via a terminal connection to your Il Matto – it is not important for this lab. If you did not get bidirectional communication between two Il Matto boards in Lab 1, ask a supervisor at the beginning of the lab and they will provide you working code.

3.1 Carrier Sense Multiple Access (CSMA) protocols

CSMA verifies the presence or absence of traffic on the shared channel before transmitting – a bit like listening to see if it's quiet before you start speaking. It is used to determine whether another transmission is in progress, by detecting the presence of a carrier signal from another transceiver, before initiating a transmission.

As presented in the ELEC3222 lectures, there are (at least) three different 'versions' of CSMA:

- **0-persistent**: when the node is ready to transmit it senses the channel and, if it is idle, transmits immediately. Otherwise, it waits for a random period of time during which it does not sense the channel, before repeating the whole process.

- **1-persistent**: when the node is ready to transmit it senses the channel and, if it is idle, transmits immediately. Otherwise, it senses the channel continuously until it becomes idle, and then transmits the message immediately. If a collision occurs during transmission, it waits for a random period of time before repeating the whole process.

- ***p*-persistent:** when the node is ready to transmit it senses the channel and, if it is idle, transmits with a probability p. If the node does not transmit (i.e. probability $1 - p$), or if the channel is not idle, it waits until the next time slot and repeats the whole process. If a collision occurs during transmission, it waits for a random period of time before repeating the whole process.

If you are unclear, refer to the ELEC3222 lecture nodes for more information.

3.2 Lab Tasks

- The rfm12_tick() function implements a simple collision avoidance mode and initiates transmissions. In the first lab, this function was called periodically in the main file in order to enable message transmission. Below, a brief explanation of how this function works:

    - It first checks the state of the transceiver. If the state is not idle, the RFM12 module is already receiving or transmitting data, and it does not want to be disturbed (hence the function returns);

    - Otherwise, if the collision flag is active, it senses the channel in order to detect a carrier. If a carrier is detected, the counter that is used to count for 'how long' the channel is free is reset. If a carrier is not detected, the counter is decremented. The function returns, and only when the counter is zero, does it start transmitting;

- Check yourself how, in the code you wrote for Lab 1, transmitting works. Identify how the number of bits is calculated for the message to be sent, considering both the header (or Protocol Control Information (PCI)) and the payload (or Service Data Unit (SDU)).

- Once you have read and verified/understood in detail how this simple logic implements collision avoidance and initiates transmissions, you have to modify this approach extracting the right functionality in order to first implement a 0-persistent CSMA protocol, and then either 1-persistent or *p*-persistent CSMA.

- The new approach should call the function, from the main file, just once – not like rfm12_tick() – and you should have two different functions: one for 0-persistent CSMA and the other for 1-persistent CSMA (or *p*-persistent CSMA depending on which you choose to implement).

- Test the code to make sure it's working. How do you know that it's working and actually doing (exactly) what it's supposed to?

If you have enough time, you can also attempt to implement the following functionality:

- If the channel is very busy (and CSMA is repeatedly detecting that the channel is not idle or collisions are occurring frequently – you decide what 'repeatedly' and 'frequently' means!), your system automatically changes the communication channel.

- To change channel, you will need to use function(s) in the RFM12B library (i.e. the physical layer) that you investigated in the last lab.

- Remember, to do this successfully, not only does the transmitter need to change channel, but also the receiver. How does this negotiation happen automatically? What happens if data is lost during negotiation?

---

4 Resources

You can find more information about RFM12B-S2 on-line:

- https://www.sparkfun.com/products/12031

The following documents are also useful resources:

- RFM12B-S2 Datasheet:
  - http://cdn.sparkfun.com/datasheets/Wireless/General/RFM12B.pdf

- Programming guide:
  - https://www.sparkfun.com/datasheets/Wireless/General/RF12B_code.pdf

- RFM12B IC Datasheet:
  - https://www.sparkfun.com/datasheets/Wireless/General/RF12B-IC.pdf