A course on Wireless Sensor Networks (WSNs)

Luis Sanabria, Jaume Barcelo

February 6, 2013

Contents

Chapter 1

About the course

1.1 Course Data

Code: 21754

Course name: "Xarxes de Sensors Sense Fils"

Teacher: Luis Sanabria and Jaume Barcelo

Credits: 4

Year: 3rd or 4th year (optional)

Trimester: Spring

1.2 Introduction

The reduction in price and size of computing and wireless communication platforms over the last years opens a new possibility for gathering and processing information: Wireless Sensor Networks. A wireless sensor node is an electronic device of small dimensions that gathers measures from the environment and transmit the data wirelessly. In wireless sensor nodes, communication is often established with other wireless sensor

nodes to exchange or pass information. It is common to have this data directed to an special device that gathers all the data and is called the network sink. As wireless sensor nodes are often battery-powered, energy saving is a relevant issue in these networks.

What follows is an extract of the first pages of [11].

> Wireless Sensor Networks (WSNs) are a result of significant breakthroughs on wireless transceiver technology, the need of event sensing and monitoring. One might think of a WSN as the skin of our bodies; apart from its importance on many other subjects, our skin senses events nearby it, like touch, temperature changes, pressure and so forth. These events are generated by an external entity, the nerves or sensors of our skin are capable to react to such events and transmit this information to the brain.
>
> There are enormous differences among characteristics of WSN and the skin, but the example given above will work as head start to understanding the technology. For instance, our skin sends the sensed event information towards the brain through the nerves, we could safely relate this medium to a wired network infrastructure. While in WSN, as its name suggests sends the sensed data towards a central node (Sink) via a wireless medium. Because of the limited radio range of each node, the route to the Sink is generally composed of jumps through different nodes (which is called a multihop route).
>
> The majority of wireless nodes in a WSN are very constrained devices due to the restrictions in

costs and sometimes harsh environments where these networks are developed. These constrains go from cost, processing power, memory, storage, radio range, spectrum and, more importantly, battery life. One of the most popular low-end nodes model, the TelosB, is equipped with 16 MHz CPU, very small flash memory (48 KB avg.), about 10 KB of RAM and works on the very crowded 2.4 GHz spectrum at rates around 250 Kbps. These limitations force WSN engineers to design applications capable of working with low processor-intensive tasks and powered with limited battery (usually two AA batteries).

Many WSN applications process the sensed event before sending the data, this processing tries to reduce the information to send. As mentioned in [1], it is less energy consuming to process one bit of information than sending it. WSN protocols and applications are tailored to power conservation rather than throughput, mainly due to cost, dimension, processing and power constraints.

WSNs may contain different kind of sensors that help monitor metrics related to: temperature, humidity, pressure, speed, direction, movement, light, soil makeup, noise levels, presence or absence of certain kinds of objects, mechanical stress and vibration. Also further information like node location can be derived from a Global Positioning System (GPS) device embedded at each node.

Because of the variety of measures than can be monitored with these small and (generally) cheap devices, a wide range of applications have been de-

veloped; the authors of [1] divide them in: military, environmental, health, home and industry applications.

- *Military Applications:* one of the first applications of WSNs. The main advantages in this area are the fact that the deployment of low cost sensors (that are subject to destruction in a battlefield) proposes a cheaper approach to sensing different types of metrics, which in turn brings new challenges to WSN applications (increased power and processing constraints). Some of the applications are related to: monitoring the movement of troops, equipment and ammunition, battlefield surveillance, terrain reconnaissance, damage assessments, snipper detection [8], [9] and threat detection, as in the case of biological, radiological or chemical attacks.

- *Environmental Applications:* most of these applications are related to animal tracking, weather conditions and threat contention [10], [12].

- *Health Applications:* a great deal of these applications are dedicated to monitor patients inside hospitals and provide them with better care. This is achieved by tracking the patients vitals or other information of interest and making it available to doctors at any time from anywhere securely through the Internet.

- *Home Applications:* technology is making its way inside our homes from various fronts, and WSN are no exception. Sensor nodes inside domestic devices will result in an increased interaction among them and allow access via the Internet. Theses applications are of great importance in fields like domotics towards a smart home/work environment. Home surveillance and multimedia WSNs for home environments are also a growing field of research.

- *Industrial Applications:* historically the monitoring of material fatigue was made by experts introducing the observed situation inside PDA devices to be collected on a central site for processing. Further sensing techniques were developed on the form of wired sensors; nevertheless its implementation was slow and expensive due the necessary wiring. WSNs bring the best of both methods by sensing the events without the need of expert personnel and the cost of wiring.

- Other implementations as mentioned in [1] are: inventory management, product quality monitoring, smart offices/houses; guidance in automatic manufacturing environments, interactive museums, factory process control and automation, machine diagnosis, transportation, vehicle tracking and detection, spectrum sensing for cognitive radio networks, underground and underwater monitoring.

1.3 Syllabus

- Lectures
    1. Introduction to WSNs.
    2. Arduino Platform.
    3. Xbee and Xbee explorer. AT commands.
    4. Xbee API mode.
    5. A sensor network with Arduino.
    6. A sensor network without Arduino.
    7. Publishing sensed data
    8. Invited talk
    9. Quiz
- Labs and seminars
    1. Blinking LED (Dimming optional)
    2. Blinking LED with push-button (dimming optional)
    3. Xbee chat
    4. Wireless doorbell
    5. Romantic light
    6. Sensor network with Arduino
    7. Sensor network with Xbee in API mode
    8. Sleeping and actuating
    9. Uploading sensed data to the Internet

1.4 Bibliography

Most of the lab assignments follow the book that you can find at the university library:

Robert Faludi "Building Wireless Sensor Networks" ([6]).

1.5 Evaluation Criteria

The grading is distributed as follows:

- Quiz, 10%
- Each lab assignment, 10%

It is necessary to obtain a decent mark in all the different evaluation aspects. To pass the course, 50 out of the total 100 points need to be obtained.

1.6 Survival guide

1.6.1 Questions and doubts

WE like to receive questions and comments. Normally, the best moment to express a doubt is during the class, as it is likely that many people in the class share the same doubt. If you feel that you have a question that needs to be discussed privately, we can discuss it right after the class.

1.6.2 Continuous feedback

At the end of the lecture, we will ask you to anonymously provide some feedback on the course. In particular, I always want to know:

- What is the most interesting thing we have seen in class.
- What is the most confusing thing in the class.
- Any other comment you may want to add.

  In labs, I will ask each group to hand in a short (few paragraphs) description of the work carried out in class,

and the members of the group that have attended the class. Note that this is different from the deliverables, which are the ones that are actually graded.

1.6.3 How to make you teachers happy

Avoid speaking while we are talking.

Chapter 2

Introduction to Arduino

2.1 Open Hardware

"There's a fine line between open source and stupidity", says Massimo Banzi to a reporter from Wired Magazine while having dinner at a restaurant in Milan.

Banzi is the man behind Arduino, an open hardware platform. The open about it relates to the fact that the device's manufacturing schematics, programming language and software development environment are free and open source. This basically means that everyone interested on building hardware-coupled solutions may take an Arduino board's schematics, modify it at will, send the new design to a China manufacturer and get the final product back home for around €10 [5].

Open hardware is supported by a variety of available licenses (like open software with LGPL, GPL, Copyleft, and others) that ensure that the protected platform can be copied, enhanced and even sold, but always recognizing the original authors. It also ensures that the resulting products are open as the original.

2.2 The Arduino Platform

Arduino was developed to teach Interaction Design [3], that meant that it required the ability to sense the surroundings and do something about it.

The platform is equipped with simple digital and analog input/output interfaces, that can be programmed to sense or react to some events. Figure 2.1 shows the Arduino Duemilanove board.

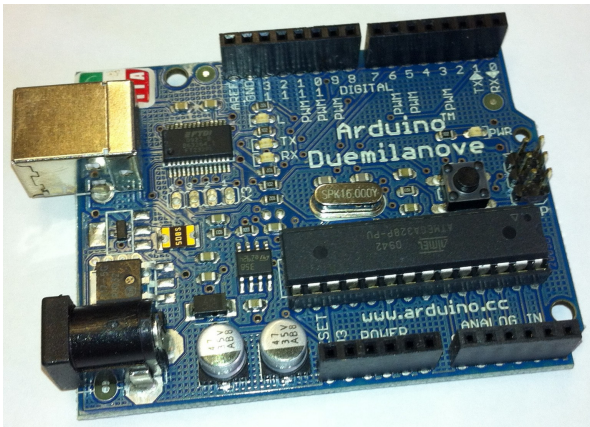

Figure 2.1: Arduino Duemilanove board

There are numerous sensors and actuators that work with Arduino. In relation to sensors: temperature, air pollution, light, GPS modules and sound are among the popular; as LEDs, speakers and digital/analog outputs are common actuators. Also, interfaces like buttons can be programmed and used as a human interactive input.

The design and electrical components of the Arduino board are available for anyone [2]. Figure 2.2 shows the connections layout of the Duemilanove model (compare with Figure 2.1).

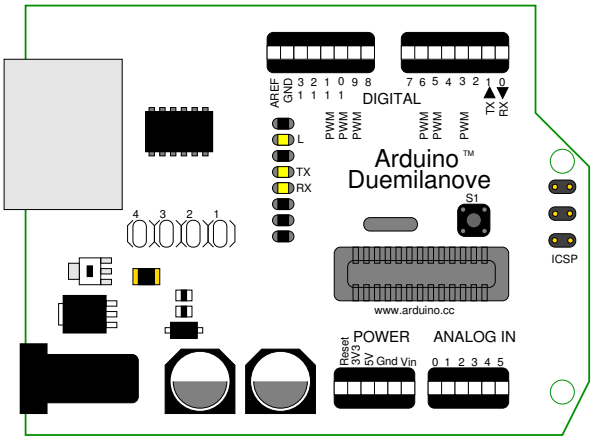

Figure 2.2: Arduino Duemilanove board: layout

2.3 Practice: Installing the Arduino IDE

In the following practice, you will spend some time getting to know the Arduino platform, its connections and how to interact with it through a PC.

2.3.1 Reviewing the hardware

As you were able to see in Figure 2.2, the Arduino board contains a whole computer on a small chip, although it is at least a thousand times less powerful.

Taking a closer look at Figure 2.2, you will be able to see *14 Digital IO pins (pins 0-13), 6 Analogue IN pins (0-5) and 6 Analogue OUT pins (pins 3, 5, 6, 9, 10, and 11).*

The *Digital IO* pins, as the name suggests can be set to input or output. Their function is specified by the sketch you create in the IDE (more on IDE in Section 2.3.2). The *Analogue IN* ports take analogue values (i.e., voltage readings from

a sensor) and convert them into a number between 0 and 1023. As for the *Analogue OUT* ports, are actually digital pins that can be reprogrammed for analogue output using the sketch you can create in the IDE.

2.3.2 The Arduino IDE

The Arduino Integrated Development Environment (IDE) is the responsible for making your code work in the Arduino board. Without entering in much unnecessary detail, what the IDE does is to translate your code into C language and compile it using `avr-gcc`, which makes it understandable to the micro-controller. This last step hides away as much as possible the complexities of programming micro-controllers, so you can spend more time thinking on your actual code.

You can download the Arduino IDE *from here*. If you are using *Linux* or *Windows* operating systems, just double click the downloaded file. This will open a folder named *arduino-[version]*, such as *arduino-1.0*. Place the folder wherever you want in your system. On the Mac, just double click the downloaded file, this will open a disk image containing the Arduino application. Drag a drop the application icon to your Applications folder.

Do not open your installed application yet. First you must teach your computer to detect the Arduino hardware through the USB ports.

Configuring the USB ports for detecting the Arduino

In Linux and OS X, the USB controllers are the same used by the operating system.

On the Mac, plug the Arduino into an USB port.

The PWR light on the board should come on. Also, the

LED labelled "L" should start blinking.

Then, a pop-up window telling you that a new network interface was found should appear. Proceed clicking "Network Preferences...", and then "Apply". Although it may appear with a status of "Not Configured", the Arduino is ready for work.

Windows machines, plug your Arduino and the "Found new Hardware Wizard" will appear. After the wizard tries to find the driver on the Internet, you will be able to select "Install from a list or specific location" button. Choose it and click next. You will be able to find the drivers under the "Drivers" folder of the Arduino Software download.

Once the drivers are installed, you can launch the IDE and start using Arduino.

Identifying the port connected to the Arduino

In the case of the **Mac**, once in the Arduino IDE, select "Serial Port" from the "Tools" menu. Select `/dev/cu/.usbmodem`; this is the name that your computer uses to refer to the Arduino board.

For **Windows**, under the operating system "Start" menu open the "Device Manager" by right-clicking on "Computer" (Vista) or "My Computer" (XP), then choose properties. On XP, click "Hardware" and choose Device Manager. On Vista, click "Device Manager".

Look for the Arduino device in the list under "Ports (COM & LPT)". Your device name will be followed by a port number, usually "COM#", where # refers to a number.

Once you have identified the COM port number for the Arduino connection, you can select that port from the Tools > Serial Port menu in the Arduino IDE.

Now the Arduino IDE can talk with the Arduino board

and program it.

What's the deal with Linux users?

As mentioned before, IDE uses the same USB controllers than Linux. So, in order to effectively detect your Arduino in Linux, simply connect it to your PC, open a Terminal a type `ls dev/ttyUSB*`. This will display all available ports. Your Arduino serial port will probable be something like `/dev/ttyUSB0`.

2.4 Practice: Blinking LED

In the following practice you will write your first Arduino application. Although simple, mastering it will provide you with clear understanding of the IDE and the components that conform the Arduino platform.

It consist of a simple code that will turn on/off LED(s) plugged to the digital IO ports of the Arduino.

2.4.1 Preparing your development environment

For Practice 2.4, you will need:

- as many LEDs as you want, but always less than the number of digital IO ports.
- a USB cable to connect your Arduino board to the PC.
- the Arduino IDE, up and running.

Turn your Arduino on by plugging it to the PC. Make sure you have selected the appropriate COM port, as it is explained in Practice 2.3 according with your operating system.

2.4.2 The code

Once inside, enter the following code:

```
1  const int LED = 13;
2
3  void setup()
4  {
5      pinMode(LED,OUTPUT);
6  }
7
8  void loop()
9  {
10     digitalWrite(LED, HIGH);
11     delay(1000);
12     digitalWrite(LED, LOW);
13     delay(1000);
14 }
```

Listing 2.1: Blinking LED example code

As you might be able to see, the code is completely readable. Let's review it line by line.

- Line 1: `const int LED = 13`, assigns the value 13 to a interger variable, named LED. In this case, this number corresponds to the digital IO port #13.
- Line 3: `void setup()` is the name of the next block of code. It is very similar to functions in languages like C/C++ and it is generally used to assign variables to ports, as well as their role.
- Line 5: `pinMode(LED,OUTPUT)` tells the Arduino how to set the pins. In this case, pin LED (#13) is set up as

an OUTPUT. `pinMode` is a function, and the words or numbers inside the parenthesis are its arguments.

- Line 8: `void loop()`: is where you define the behaviour of your device. The statements contained in `loop()` are repeated over and over again until the device is turned off.

- Line 10: `digitalWrite(LED, HIGH)` works as a power socket for pins. In this case, the command is indicating to turn pin `LED` into `HIGH`, which instructs Arduino to turn the output pin to 5V. If you have connected a LED in this pin, the result is that it will turn on (hopefully). Turning on and off the pin allow us to see what the software is making the hardware do; the LED is an actuator.

- Line 11: `delay(1000)` tells the processor to wait for 1000 *milliseconds* before proceeding to the next code line.

- Line 12: `digitalWrite(LED, LOW)` as with Line 10, this function turns pin `LED` to 0V, causing the connected LED to turn off. You can do a mental map in which $HIGH \rightarrow ON$, $LOW \rightarrow OFF$.

- Line 13: because the last instruction was to set the LED off, this will keep it that way for an additional 1000 *millisecods*.

To see your work, just insert the longer leg of the LED into the digital IO port you assigned to variable `LED` on your code (digital pin 13), and the shorter leg to ground (GND). Figure 2.3 shows the desired layout.

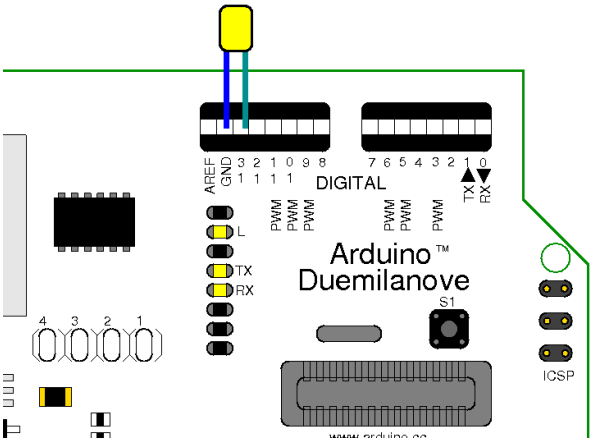

Figure 2.3: Blinking LED layout

2.5 Practice: Blinking LED Advanced

It will be very boring to just have a blinking LED. That is because in this practice we will be incorporating some hardware and software tweaks that will allow us to have a little more control over the LED. Or let's say, we will make a basic lamp.

What we want to prototype is a LED that turns on or off whenever we press a bottom. Before we dwell into detail, let's review what we will need:

- A breadboard (we will be using Figure 2.4 as a guide).
- Wire to tie together the different parts of your circuit.
- One 10K Ohm resistor.
- One pushbutton switch.

Breadboards will help us to build circuits. It allows for effective connection between components without worrying about

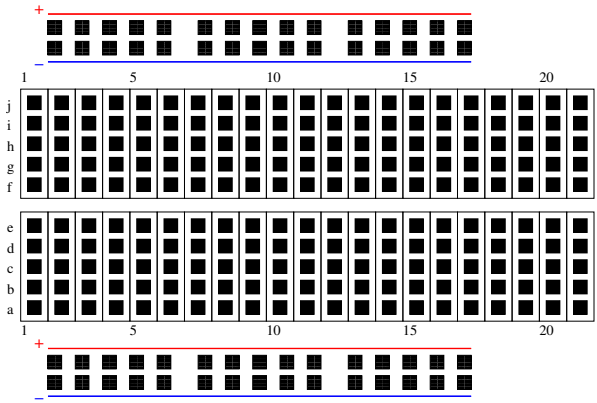

Figure 2.4: Breadboard

the electrical subtleties or hazards. Taking Figure 2.4 as a reference, the breadboard has internal electrical connections that makes it possible to tie multiple components to a single point. It does so by representing a *physical* connection as multiple rows of the same column. That is: holes 1a and 1d are physically connected inside the breadboard's circuitry, whereas 3d and 4a are not.

Each breadboard is divided by thick spaces among different sections. In Figure 2.4, there are four distinct sections: two with the + and − symbols, and two with numbers and letters. The latter was described above, whereas the former works in the opposite way: holes are connected with other holes in the same row. This section is often used to power the circuit, but more on that further in the practice.

Before writing any code, try to assemble the parts as shown in Figure 2.5.

To avoid any confusion, let's review the layout component by component:

1. Place the pushbutton on your breadboard. In Figure 2.5, the two pushbutton "legs" are inserted into holes 5c and

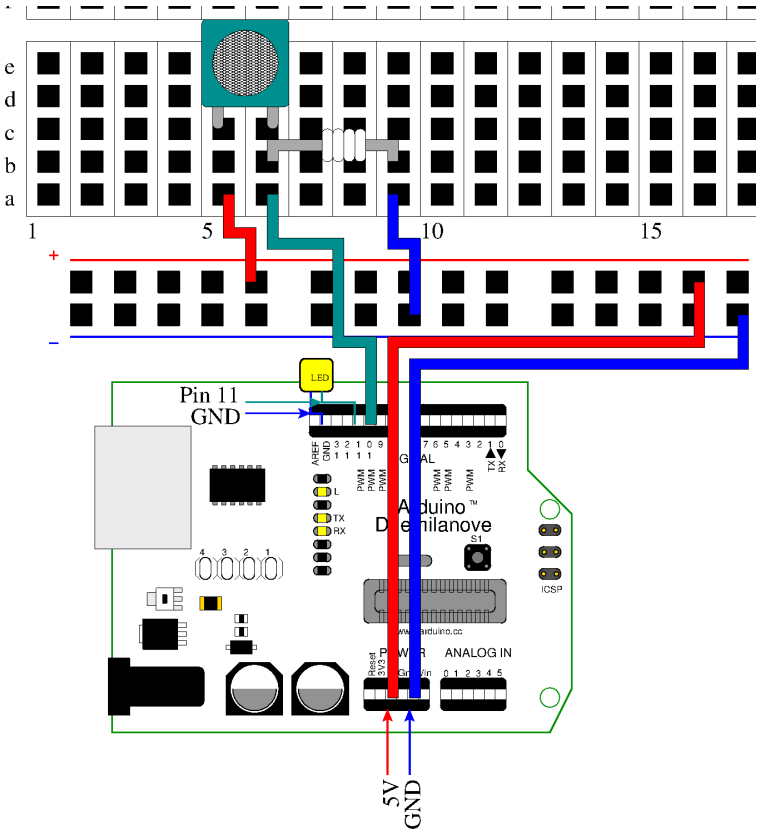

Figure 2.5: Blinking LED advanced layout

6c. In this example, the pushbutton will be energised through the 5c leg.

2. Connect one of the legs of the resistor to the negative leg of the pushbutton (hole 6b). This will physically connect the resistor to one of the legs of the pushbutton. Insert the other leg on hole 9b.

3. In order to avoid confusion, cables on all figures are color coded. *Red* represents power cables, *blue* are connections to GND and *cyan* are connections to IO pins on the Arduino. Try to duplicate the layout of Figure 2.5.

Note that connecting the + row to the Arduino's 5V pin will provide 5V to all the + row. The same is true for GND and the − row. This is very useful to avoid running out of 5V of GND pins.

2.5.1 The code

Type the following instructions as a new file in the Arduino IDE:

```
1  // Turns on LED when pushbuttom is pressed and
2  // turns it off when pressed again.
3
4  const int LED = 11;
5  const int BUTTON = 10;
6  int val = 0;
7  int old_val = 0;
8  int state = 0;
9
10 void setup(){
11     pinMode(LED, OUTPUT);
12     pinMode(BUTTON, INPUT);
13 }
14
15 void loop(){
```

```
16      val = digitalRead(BUTTON);
17
18      //check if the button was pushed
19      if((val == HIGH) && (old_val == LOW)){
20          state = 1 - state;
21          delay(10);
22      }
23
24      old_val = val;
25      if(state == 1){
26          digitalWrite(LED, HIGH);
27      }else{
28          digitalWrite(LED, LOW);
29      }
30  }
```

Listing 2.2: Blinking LED advanced example code

Let's review the code:

- Line 4: sets the pin for the LED.
- Line 5: assigns the input pin where the pushbutton is connected.
- Line 6: `val` is the variable holding the state of the input pin corresponding to the pushbutton.
- Line 7: `old_val` holds `val`'s previous value.
- Line 8: the variable `state` determines de condition of the LED. 0 = off and 1 = on.
- Line 11: the function `pinMode()` sets the roll of each pin. In this case, pin `LED` is set to `OUTPUT`.
- Line 12: sets pin `BUTTON` to `INPUT`.
- Line 16: asks whether there is any power at the specified pin. It returns HIGH or LOW if the button is being pushed or not, respectively.

- Line 19: if the button is being pushed, then `val` = HIGH and `old_val` = LOW. This provokes a change in `state`.
- Line 21: prevents errors in the change of `state`. Given that `loop()` repeats several hundred thousand times per second, making the processor wait a little bit allows for a correct reading of the pushbutton.
- Line 24: the value of `val` is now old. Notice that once the LED is turned on, `val` = `old_val` = LOW. Furthermore, `val` only changes when the button is pushed.
- Line 26: turn LED on.

Chapter 3

An introduction to Zigbee and IEEE 802.15.4

Zigbee and IEEE 802.15.4 are specifications for the higher and lower network layers of WSNs.[1] Zigbee devices are designed for low-range, low-complexity, low-cost and low-consumption applications. Zigbee includes low duty-cycle capabilities, which mean that the devices can sleep for most of the time. A zigbee product can potentially run on batteries for years. In our particular course, we use zigbee combined with Arduino. As Arduino is not designed for low consumption, it kills the possibility of running our prototypes on batteries for such a long time.

Zigbee defines the network and application layers of the devices and relies on IEEE 802.15.4 for the MAC and PHY layers. The specification of Zigbee standards is carried out by the Zigbee Alliance, which comprises all kind of companies (semiconductor industry, OEMs, software developers, etc.). The standardization of IEEE 802.15.4 is done by the IEEE.

[1] This chapter is based on [4] and [7]

3.1 IEEE 802.15.4

3.1.1 IEEE 802.15.4 PHY layer

IEEE 802.15.4 operate in three different bands: 868 MHz (1 channel), 915 MHz (10 channels) and 2.4 GHz (16 channels). The 2.4 GHz band is a band for industrial, scientific and medical uses (ISM). This band is shared by many technologies, including the IEEE 802.11 family. Spread spectrum techniques are used to alleviate the problem of interference. The modulations that are used are BPSK, ASK and O-QPSK. Direct sequence spread spectrum (DSSS) or parallel sequence spread spectrum (PSSS) is used.

The functionalities offered by the PHY layer of IEEE 802.15.4 include link quality estimation, energy detection and clear channel assessment.

3.1.2 IEEE 802.15.4 MAC layer

Regarding the MAC layer, the standard differentiates two types of devices: Reduced Function Devices (RFD) and Full Function Devices (FFD). RFDs are simpler, which means that their power consumption and their price can be lower. The only role that RFDs can play in a sensor network is that of end devices. RFDs cannot forward packets of other nodes.

The FFD have extended functionality and can act as end devices and also as network coordinators. Network coordinators can send beacons for synchronization purposes and provide network join services. FFDs can also relay packets of other nodes for multi-hop communication.

The standard considers two different alternatives for nodes to communicate: master-slave and peer-to-peer. In the master-slave situation, the slave communicates only with its master.

There is a WSN coordinator (which has to be a FFD) that decides which of the two alternatives is used.

In the master-slave situation, the master periodically broadcasts a beacon and the slave has to synchronize to the beacon to communicate with the master. The beacon contains control information about the WSN and also a superframe structure. The superframe divides the time interval between two beacons in three different parts: The Contention Access Period (CAP), the Contention Free Period (CFP) and the inactive period.

In the CAP, the sensors contend for channel access using slotted CSMA/CA. In the CFP, some of the nodes are assigned Guaranteed Time Slots (GTS) by the coordinator. This means that those slots are reserved for a particular sensor, and that sensor can transmit without contending for the channel. Finally, in the inactive period, the all the nodes can sleep to save power.

The slaves only have to wake up to receive the beacons and to transmit or receive data. The master uses the beacon to indicate which of the slaves have data to be delivered.

3.2 Devices

There are two different types of devices: full-function devices (FFD) and reduced-function devices (RFD). FFDs are more powerful and can assume any role in the network. RFDs are simpler, and can assume only the role of "Device" in the IEEE 802.15.4 terminology (equivalent to "ZigBee End Device" in the ZigBee terminology.

The other two possible roles, that can be assumed only by FFDs are "Coordinator" and "PAN Coordinator" in the IEEE 802.15.4 terminology. These are equivalent to "ZigBee Router" and "ZigBee Coordinator" in the ZigBee terminology.

A ZigBee End Device can be the source or destination of a packet, but it cannot forward packets for other nodes. A ZigBee Router can relay packets of other nodes. And a ZigBee Coordinator is the head of the network. In every every network, there is one and only one Coordinator.

Regarding the topologies, we can consider three different cases: star, tree and mesh.

3.3 Channel Access

In IEEE 802.15.4 there are two methods for networking: beacon-enabled networking and non-beacon networking. In beacon-enabled networking the coordinator transmits periodical beacons to synchronize the network. In these beacons, it is possible to define a super-frame structure. The super-frame structure spans the channel time between two consecutive beacons, and differentiates three periods: Contention Access Period (CAP), Contention Free Period (CFP, optional) and Inactive Period (IP, optional). The time is divided in slots, and multiple slots can be assigned to each of the different periods. In the CAP, the devices use CSMA/CA to contend for channel access. In the CFP, the channel time is reserved and only the devices that own the reservation can transmit. In the IP, the devices can go to sleep to save energy.

In non-beacon networking, the devices must always contend for the channel. The contention uses the CSMA/CA protocol. Before transmitting, the devices sense the channel for the presence of an ongoing transmission. If a transmission is detected, the devices backs off and re-attempts after a random period of time.

There are two mechanism to detect an ongoing transmission. Energy detection in which the device simply measures the amount of energy on the channel, and carrier detection in

which the device looks for the presence of an IEEE 802.15.4 carrier. Either one or the other, or even a combination of both, can be used.

3.4 MAC handshake

In a beacon-enabled network, a node that wants to transmit data to the coordinator waits for the CFP if it has a reservation. Otherwise, it waits for the CAP. Then it transmits the packet (using CSMA/CA if it is in contention mode). The sender can request acknowledgement and, in this case, the coordinator may acknowledge the reception.

When a coordinator wants to transmit data to a device it indicates in the beacon the destination of the packet. The device processes the beacon and learns that there is data pending for it. After that, the device sends a "data request" message to the coordinator that the coordinator must acknowledge. Then the coordinator sends the data packet to the device which may acknowledge the correct reception.

In a non-beacon network, a device can transmit to the coordinator whenever the channel is sensed empty. If a coordinator has data for a device, it waits until the device sends a "data request" which must be acknowledged. Then the coordinator sends the data to the device, that may acknowledge the correct reception.

3.5 MAC problems in multi-hop wireless networks

There are two well-known problems that recurrently appear in multi-hop wireless networks. The first one is the "hidden node problem" and is represented in Figure 3.1. Imagine that

nodes A and C cannot carrier sense each other. Then it can happen that C starts a transmission while A is transmitting to B. This results in a collision and the receiver may not be able to recover any of the packets.

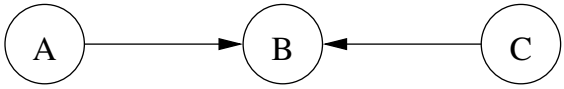

Figure 3.1: Hidden node problem

The other problem is the "exposed node problem", illustrated in Figure 3.2. In the example, E is transmitting to D. Node F wants to transmit to G, but it senses the channel busy (as it detects E's transmission). F does not transmit when, in fact, it could transmit without interfering with the other transmission.

Figure 3.2: Exposed node problem

3.6 Addresses and Network Layer

Each device has two addresses: a 16 bit short address and a 64-bit extended address.

Chapter 4

Introduction to XBee

One of the main characteristics of WSNs is the ability each node has to wirelessly communicate with other nodes. During this course we will be doing this with ZigBee protocol compliant radios, like XBee [6].

Throughout this section you will be introduced to the different components and code that will allow you to set a basic wireless network with XBee modules.

4.1 The XBee module hardware configuration

XBee modules come in different configurations. The one we will be using is called XBee Series 2 with wire antenna as it is shown in Figure 4.1.

This device supports different kinds of ZigBee in mesh networking. Its wire antenna provides omnidirectional coverage, or what is the same as saying that its coverage is pretty much the same in all directions when the antenna is straight and perpendicular to the module.

Figure 4.1: XBee Series 2 with wire antenna

If you flip the XBee, you will be able to see the pins through which it can send/receive data to/from sensors, communicate with Arduino, connection to a power supply and GND (more information about the pins can be found in page 15 of [6]).

Preparing the XBee for configuration

We can access and program the XBee through any terminal application and a USB connection. The *breakout board* shown in Figure 4.2 allows us to: 1) plug the XBee into a breadboard, facilitating the wired connections with other components (including the Arduino); as well as the ability to 2) establish a USB connection to configure the XBee.

As the pins on the XBee are separated differently than the holes in the breadboard, every time a configuration or wired connection is needed, the XBee should be placed in the breakout board as shown in Figure 4.3, and then placed on the breadboard.

It is important to notice that once the XBee is placed on

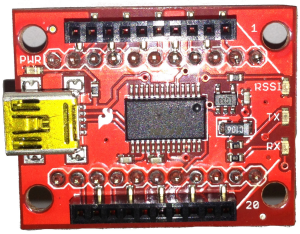

Figure 4.2: XBee Explorer board from SparkFun

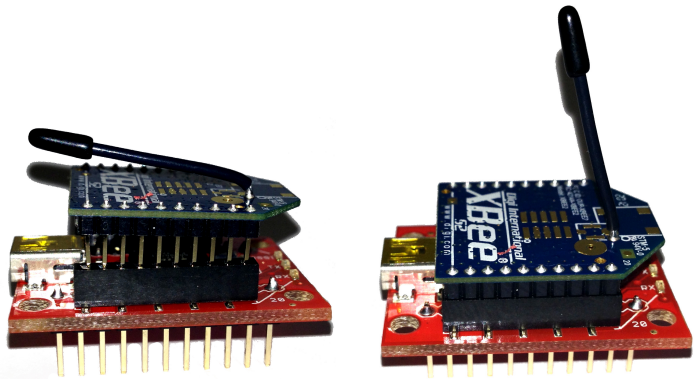

Figure 4.3: XBee and breakout board: Left: XBee outside; see the different spacing of the pings. Right: XBee inside; setup for configuring and pluging into breadboard.

the breakout board, the pins functions change. The new role of each pin is now the displayed underneath the breakout board.

Now that the device is properly placed, we need to setup a connection so the current configuration can be reviewed and changed.

Accessing the firmware

The XBee has a microcontroller running a configurable firmware. This firmware holds the necessary information for addressing, communication, security and utility functions. You can configure this firmware to change different settings like: local address, security settings, destination address and how the analog sensors connected to its pins are read.

As for now, the official way to update this firmware is through a program called *X-CTU* and can be downloaded for free from the *XBee manufacturer's website*.

X-CTU is only available for the Microsoft Windows operating system, nevertheless you have the virtualization option in OS X, as well as WINE Windows emulator in Linux.

To take a peek at the current XBee configuration:

1. Plug it into one of your computer's USB ports and launch the X-CTU application.

2. Select the appropriate Com Port listed under *Select Com Port*. This port should be the same in which your XBee is connected.

3. Confirm that everything is setup correctly by clicking on the *Test/Query* button. If everything is alright, a pop-up window will display the modem type and firmware version.

4. Change to the *Modem Configuration* tab on the top the X-CTU window. This tab will show you how the firmware is configured.

5. Under *Modem Parameters and Firmware*, click on the *Read* button. This will fill the current window with the current firmware configuration, as well as the XBee's *Modem* and *Function Set*.

Luckily, X-CTU is only required for upgrading the firmware. For changing the XBee's configuration we only need a USB connection and a terminal software. Nevertheless, this is only possible if the XBee is on *AT command mode* (capable of receiving human commands and forward messages without performing any modification, see Table 4.1). To set it on, follow these steps:

1. In the *Model Configuration* tab of the X-CTU, check that the *Modem type* is set to `XB24-ZB`.

2. To start, we are going to configure two XBee radios. Under *Function Set*, choose `ZIGBEE COORDINATOR AT`.

3. Choose any version greater than `0x2070`.

4. Click on the *Write* button to program this device as a coordinator.

Once the installation is complete, gently remove the USB from the first XBee radio a plug it into another. Repeat the process described above, but now under *Function Set* choose `ZIGBEE ROUTER AT`. Select the highest version available and click on *Write* to program the device.

It is important to distinguish between the two XBee you just configured, given that they behave differently. Every ZigBee network must contain only one coordinator radio, this

way the network can be properly defined and managed. Mark which configuration each radio has with a sticker to eliminate any confusion.

Table 4.1: XBee AT modes

Transparent mode	**Command mode**
Talk *through* the XBee	Talk *to* the XBee itself
Any data can me sent through	Only responds to AT commands
Default state	+++ to enter mode
Wait 10 seconds to return to this mode	Times out after 10 seconds of no input

Configuring the XBee through a terminal

There are a lot of terminal applications. Fortunately, most of them need the same kind of information to establish a connection through USB. Table 4.2 gathers the required settings for a serial terminal software attempting to establish a connection with the XBee.

Table 4.2: Default terminal settings for establishing a connection with an XBee

Setting	**Value**
Baud	9600
Data	8 bit
Parity	None
Stop bits	1
Flow control	None
Line feed	CR+LF or Auto Line Feed
Local echo	on

To check if you are already inside the XBee, try asking the radio to go to command mode issuing the `+++` instruction. If after a moment an `OK` appears at the right hand side, then you are in!

Reviewing some AT commands for the XBee

Issuing commands from a serial connection (like the one you established with the terminal program) to the XBee follows a simple guideline: *instruction parameter* `<CR>`. Where `<CR>` accounts for *carriage return*, and just means that you have to press the Return (Enter) key to submit the command. Passing and empty *parameter* just outputs the current value of the specified register (the *instruction* part of the command).

All AT commands start with the `AT` prefix (accounting for *attention*) and then are followed by a two letter character command identification. Some of the basic AT commands are described below, as well as in [6].

- `AT`: gets the attention of the XBee. Its normal output is `OK`. If you do not receive this output, you've probably timed out of command mode and need to reissue the `+++` command to get back in it.

- `ATID`: without any parameter it shows the current Personal Area Network ID (PAN ID) that is assigned to the radio. You can set a PAN ID passing an hexadecimal number in the range 0x0-0xFFFF as a parameter.

- `ATSH/ATDL`: it shows the *high* or *low* parts of the unique XBee 64-bit serial number, respectively. This number cannot be changed, so passing a parameter will produce an `ERROR` response.

- `ATDH/ATDL`: it shows the *high* or *low* parts of the destination address the local radio will forward messages to, respectively. Putting address information after `ATDH` or `ATDL` will set the *high* or *low* parts of the destination address, accordingly.

- `ATWR`: saves the current configuration to firmware, so it will become the default configuration the next time you power on the XBee.

4.2 Practice: Simple chat with XBee

WSNs are composed of nodes able to send messages among themselves. In this practice you will be guided through the configuration of (at least) two XBees to build a basic chat application. Furthermore, you will have the opportunity to familiarize yourself with the XBee and the different AT commands described in Chapter 4.

You will need:

- One XBee Series 2 configured as a `ZIGBEE COORDINATOR AT`.
- One XBee Series 2 configured as a `ZIGBEE ROUTER AT`.
- As many breakout boards and USB cable A to mini B as XBee radios.
- One computer per XBee. It is less confusing than establishing multiple terminal sessions from one computer.

We need to be able to distinguish the coordinator radio from the router radios. It is easier if you write this addresses down. Proceed as follows:

1. Establish a terminal connection to the coordinator radio.
2. Once inside, issue the `+++` to enter to command mode.
3. Type `ATSL` to reveal the lower part of the XBee serial number.

4. Write it down: **Coordinator address:** 0013A200 _____

Repeat the same for the router AT.

Router address: 0013A200 _____

Now, let's configure the coordinator.

4.2.1 The code: coordinator

The settings for the coordinator are contained in Table 4.3 below.

Table 4.3: XBee coordinator settings for simple chat

Description	Command	Parameter
PAN ID	ATID	**2013**
Destination address *high*	ATDH	**0013A200**
Destination address *low*	ATDL	_____

Note that the *Destination address low* specified in Table 4.3 correspond to the router radio.

Issuing the commands on the terminal window will look like the listing below.

```
+++
OK
ATID 2013
OK
ATDH 0013A300
OK
ATDL ____ //put the lower part of the router address
OK
ATID
2013
ATDH
0013A300
ATDL
____
ATWR
OK
```

Listing 4.1: Coordinator settings as seen in the terminal

You will receive an `OK` after issuing a command (as in Line 4) as well as when writing to the firmware (Line 16).

4.2.2 The code: router

The settings for the router must contain the same information collected for the coordinator. Fill out Table 4.4 accordingly.

Table 4.4: XBee router settings for simple chat

Description	Command	Parameter
PAN ID	`ATID`	**2013**
Destination address *high*	`ATDH`	**0013A200**
Destination address *low*	`ATDL`	_____

Note that the *Destination address low* specified in Table 4.4 correspond to the coordinator radio.

Chat!

Now you just have to connect each XBee to one computer and establish a terminal connection to each one (or connect the two radios to the same computer running two different terminal applications, one for each XBee). Make sure all the connection settings are as specified in Table 4.2, so you will not have any problems.

If both radios are in transparent mode (see Table 4.1), everything you type in one terminal will be forwarded to the other XBee.

Bibliography

[1] I. Akyildiz and M.C. Vuran. *Wireless sensor networks*. John Wiley & Sons, Inc., 2010.

[2] Arduino Team. Arduino homepage. http://www.arduino.cc, October 2012.

[3] Banzi, M. *Getting Started with arduino*. Make Books, 2008.

[4] P. Baronti, P. Pillai, V.W.C. Chook, S. Chessa, A. Gotta, and Y.F. Hu. Wireless sensor networks: A survey on the state of the art and the 802.15. 4 and ZigBee standards. *Computer communications*, 30(7):1655–1695, 2007.

[5] C. Thompson. Build it. Share it. Profit. Can Open Source Hardware Work? http://bit.ly/ShWD43, October 2008.

[6] R. Faludi. *Building Wireless Sensor Networks: with ZigBee, XBee, Arduino, and Processing*. O'Reilly Media, Incorporated, 2010.

[7] S. Farahani. *ZigBee wireless networks and transceivers*. Newnes, 2008.

[8] Á. Lédeczi, A. Nádas, P. Völgyesi, G. Balogh, B. Kusy, J. Sallai, G. Pap, S. Dóra, K. Molnár, M. Maróti, et al. Countersniper system for urban warfare. *ACM Transactions on Sensor Networks (TOSN)*, 1(2):153–177, 2005.

[9] J.A. Mazurek, J.E. Barger, M. Brinn, R.J. Mullen, D. Price, S.E. Ritter, and D. Schmitt. Boomerang mobile counter shooter detection system. In *Proceedings of SPIE*, volume 5778, page 264, 2005.

[10] J. Polastre, R. Szewczyk, A. Mainwaring, D. Culler, and J. Anderson. Analysis of wireless sensor networks for habitat monitoring. *Wireless sensor networks*, pages 399–423, 2004.

[11] Sanabria, L. Localization Procedure for Wireless Sensor Networks. Master's thesis, Universitat Pompeu Fabra, Barcelona, 2012.

[12] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin. Habitat monitoring with sensor networks. *Communications of the ACM*, 47(6):34–40, 2004.