

Wireless Sensor Networks

Walter Tiberti

UNIVAQ-DISIM-DEWS

Contents

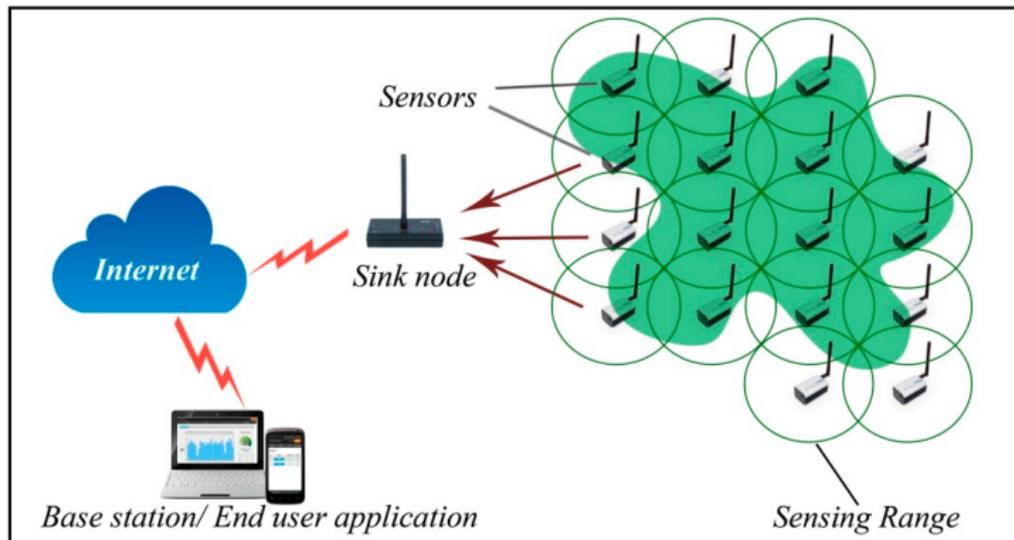
- 1 Block 1
 - Introduction
 - IEEE 802.15.4
 - MAC
 - Other standards
 - IEEE 802.15.4e & IEEE 802.15.9
 - WSN Hardware
 - WSN Software
 - TinyOS
- 2 Block 2
 - Programming WSN motes
 - Classical Embedded System Approach
 - TinyOS-based development
 - Programming using other OSES
- 3 Block 3
 - WSN Homelab

Introduction

Terminology, Architectures, Applications

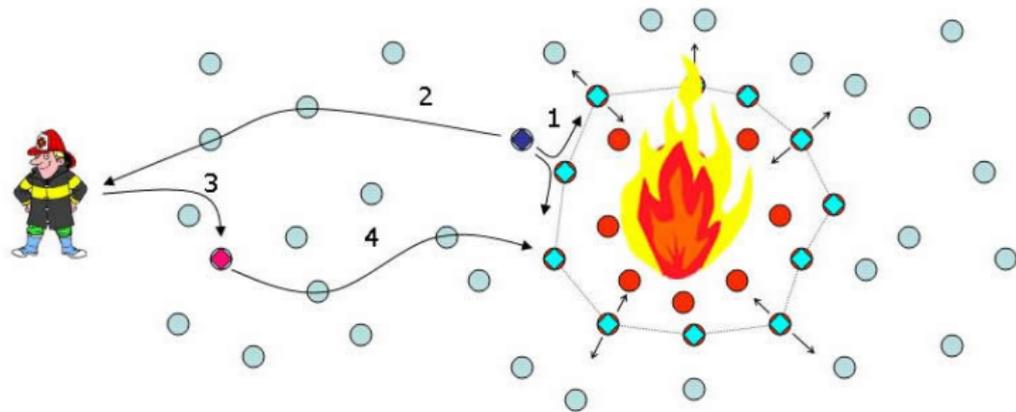
WSN : a definition

- 1 **WSN**: Wireless Sensor Network
- 2 Small ad-hoc networks composed by **small**, battery-powered, resource-constrained, **sensor-equipped** nodes (called **moten**)



WSN applications

- 1 Environmental Sensing
- 2 Healthcare
- 3 Smart agriculture
- 4 Industrial monitoring
- 5 Chemical Hazards detection
- 6 Disaster Prevention
- 7 Wireless interconnection in larger systems



WSN : common terminology

Mote a WSN node

Sink Node the node which acts as gateway of the WSN

Basestation i.e. a generic gateway device

Cluster group of interconnected motes

WSN : HW/SW overview

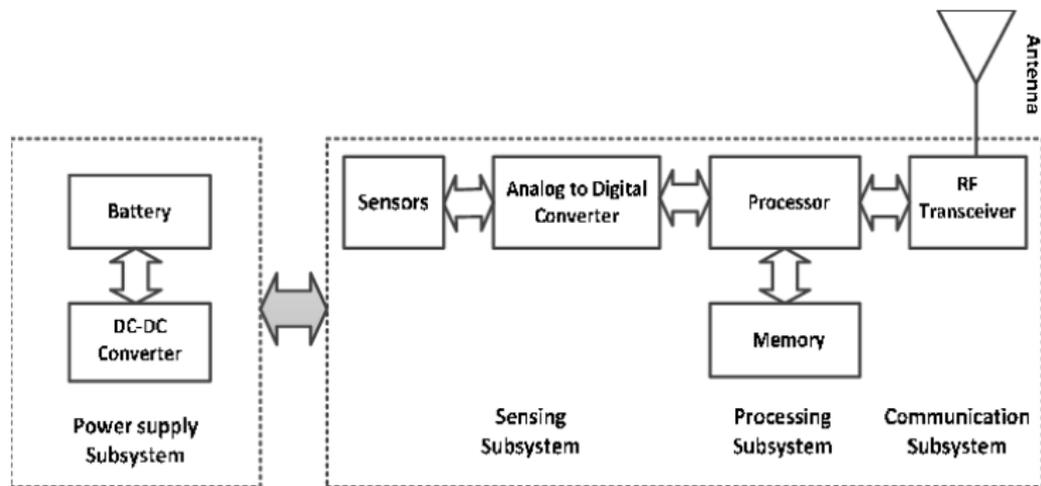
① Hardware:

- ① A small **MCU** (e.g. MSP430, AVR) with small RAM (8KB), small **Flash** storage for code (48KB) and optionally some EEPROM storage
- ② A **radio chip** (e.g. CC2420) connected via e.g. SPI and its antenna
- ③ A way for programming the node (e.g. a **USB-to-Serial** chip or an external **programming board**)
- ④ A set of on-board **sensors** or an external **sensor board**.

② Software:

- ① An optional internal **boot stage loader** (BSL)
- ② An optional **Operating System**
- ③ The node application
- ④ A **protocol stack** for the communications
- ⑤ An optional **middleware**

WSN : architecture sketch



WSN : communications

- 1 Among nodes: **Radio**.
- 2 With programmers/PC: usually via **Serial (UART) port**
- 3 With on-board peripherals: **UART, I2C, SPI**

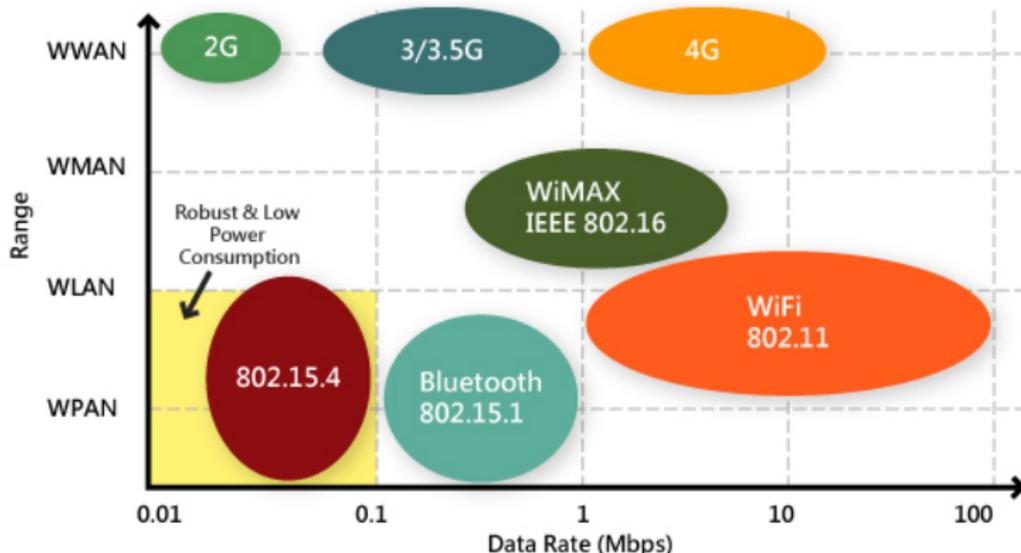
IEEE 802.15.4

Introduction, Layers

IEEE 802.15.4: target

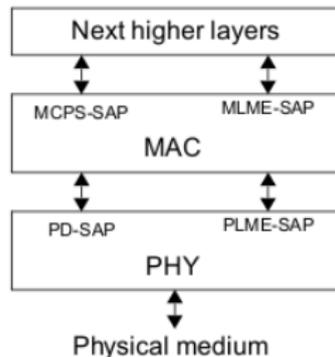
- 1 The **IEEE 802.15.4** standard is the *de-facto* communication protocol for Low-Rate Wireless Personal Area Network (**LR-WPAN**)
- 2 The last version was released in 2015
- 3 Link (from IEEExplorer)

<https://ieeexplore.ieee.org/document/7460875>



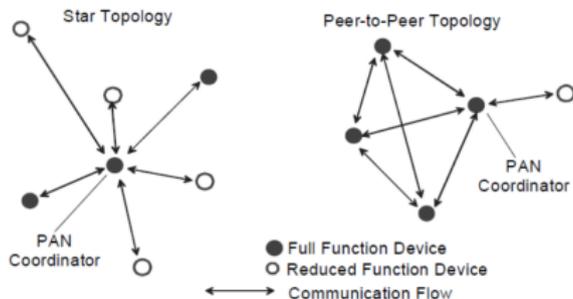
What covers?

- 1 Physical Layer (**PHY**): radio frequencies, modulations etc.
- 2 Medium Access Control Layer (**MAC**): frame format, channel access mechanisms, how to avoid collisions, message types, *security*
- 3 Other layers are outside the scope of the standard. Usually, in most application there is no need for additional layers for basic mote communications



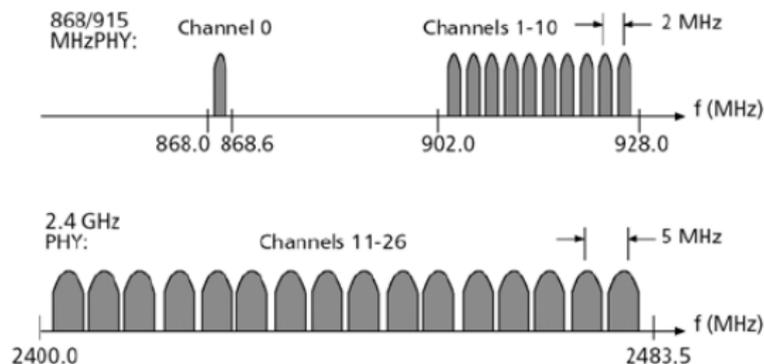
Overview

- ① Two types of node:
 - ① Reduced Function Device (RFD), i.e. all those devices which has minimal resource and computing power.
 - ② Full Function Device (FFD), i.e. more powerful devices which can act also as PAN coordinator
- ② A PAN coordinator is the device designed to manage the whole WSN synchronization
- ③ Communications types: Beacon-Enabled and Beaconless
- ④ Supported topologies:



Physical Layer (PHY)

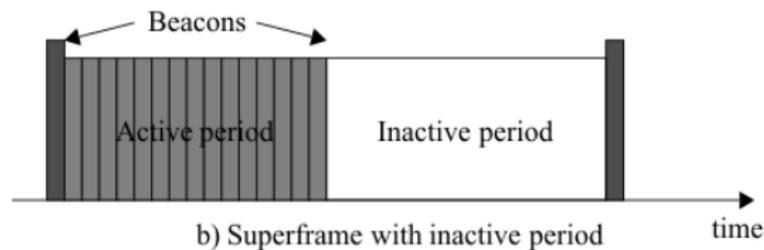
1 Frequencies/Channels:



- 2 Modulations: various, from BPSK, ASK, CSSS, Q-PSK etc.
- 3 Other features: **Link Quality Indicator**, **Energy Detection**, **Clear Channel Assessment**

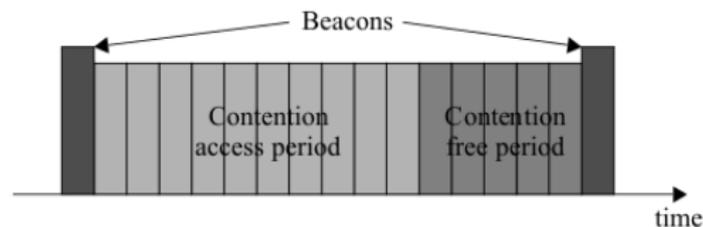
Medium Access Control Layer (MAC)

- 1 Channel Access: **CSMA-CA**, **ALOHA**, slotted in beacon-enabled PANs and unslotted in beaconless PANs
- 2 **Superframe**: timeframe between two beacons



Medium Access Control Layer (MAC)

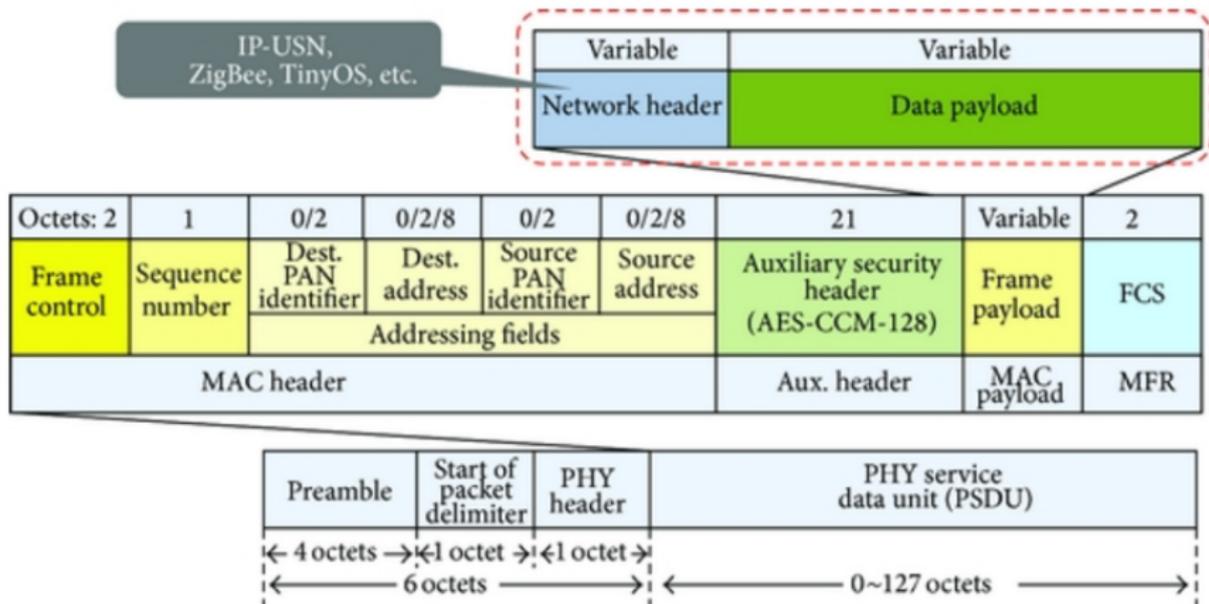
- 1 The active period can be divided into two sub-periods: the **Contention Access Period (CAP)** in which the nodes compete for the channel access and the **Contention Free Period (CFP)** in which certain nodes have reserved timeslots



MAC features

- 1 **Association**: step in which a device ask to take part into a PAN
- 2 Each device in the PAN has a **Short Address** (16 bits) and a **Long Address** (64 bits)
- 3 CRC code for data integrity checks
- 4 Optional **Acknowledgement frame** for every transmitted frame

Frame structure



Messages & Interfaces

- 1 MAC interfaces: **MLME** & **MCPS**
- 2 Each interface provides at least one of:
 - **Request** (commands, e.g. **MLME_Associate.Request**)
 - **Response**, (request success/failure)
 - **Indication** (data, e.g. **MCPS_Data.Indication**)
 - **Confirm** (command confirmation)

Security

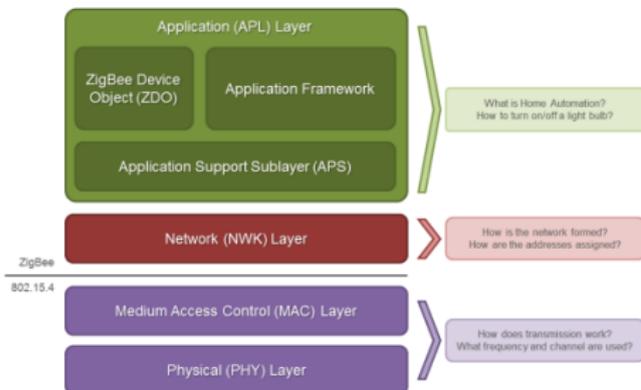
- 1 The IEEE 802.15.4 provide some basic security-oriented mechanism, e.g. Security On/Off switch, auxiliary header support etc.
- 2 However, no full specification of which mechanism to use is present

Other standards

IEEE 802.15.4e, IEEE 802.15.9

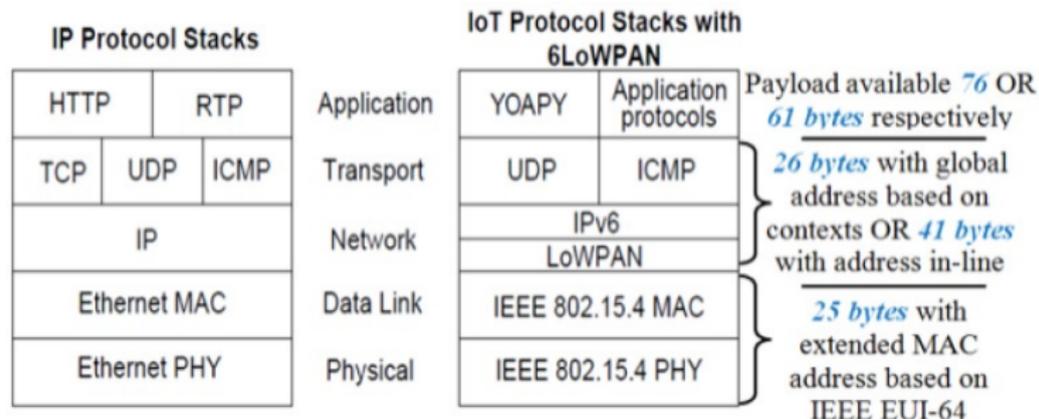
ZigBee

- Developed by the ZigBee alliance, this standard provides the missing layers to complete a full protocol stack



6LoWPAN

- ① From IEEE 802.15.4 frames to IPv6 packets via header compression



BLIP/COAP

- 1 **BLIP**: a lightweight message protocol
- 2 **CoAP**: (Constrained Application Protocol) Web transfer protocol for IoT and Machine-To-Machine applications

IEEE 802.15.4e

- 1 The **IEEE 802.15.4e** is an extension to the IEEE 802.15.4 standard which defines new MAC layers specific for industrial environments
- 2 Example feature: **Frequency Hopping**
- 3 It has been added to the IEEE 802.15.4 specification from the 2011 version

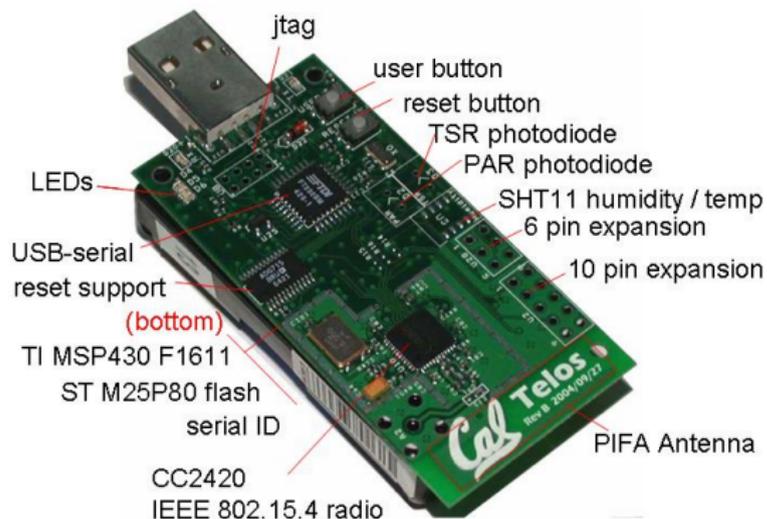
IEEE 802.15.9

- 1 Security-oriented protocol: defines a set of cryptographic keys transport mechanisms to be used in IEEE 802.15.4 networks
- 2 Features: New layer (MPX), Multiplexing, Fragmentation support, Key Management via a Key Management Protocol (KMP)

WSN Hardware

Platforms

Example 1: *telosb*



Example 2: *MicaZ*



Sensor & Programming boards

mib520 programming board and mst300 sensorboard



WSN Software

OSes and Frameworks

bare-metal

- 1 If the application is extremely simple, it is possible to develop software applications directly without any third-party framework/library/OS support
- 2 In this context, the datasheet of both the MCU, the radio chip and the board schematic are the only information needed to develop an application
- 3 Keep in mind that the final objective is to obtain a file containing the machine code which the programmer device can read and upload on the mote
- 4 An example will be presented later

TinyOS (1/5)



- 1 **TinyOS** is a framework for creating robust WSN applications
- 2 It provides a **platform-independent** software layer which allow developers to avoid dealing with the heterogeneity and the low-level hardware details of each platforms
- 3 Often, it is referred as an *Operating System* although it is not properly an OS
- 4 It is open-source (GPL)
- 5 Current release version is **2.1.2**

TinyOS (2/5)

- 1 TinyOS (and all the application using it) is written in the **NesC** language, which is a dialect of the C programming language
 - It adds a series of primitives for enabling an **event-based** programming style
 - **Component** = **Configuration** + **Module**
 - Multiple components are *connected* though **interfaces**. Each components defines a set of interfaces it **provides** and a set of interfaces it **needs** for working.
 - The details on how components and their interfaces are connected are listed in the **configurations** of the components

TinyOS (3/5)

- 1 Interfaces contains **Commands** and **Events**
 - Commands are implemented in the components which provides the interface, while they can be **called** by the components who use the interface (**call** keyword)
 - The events are **signaled** (**signal** keyword) by the component which provide the interface and must be implemented in the components which uses it
- 2 **Split-phase operation**: a particular operation which is carried out by a sequence of a command and an event response. Example:
A.startRadio() command → **A.radioStarted()** event
- 3 **Tasks**: a lightweight multi-threading-like primitive. A task can be used to perform time-taking operations in background

TinyOS (4/5)

- 1 TinyOS support a large number of platforms, MCUs, radio chips and sensorboards
- 2 It also features a set of common algorithms and peripheral-support drivers (e.g. radio chip drivers)
- 3 Also, a set of example, tutorial and documentation resource can be found directly in the TinyOS distribution

TinyOS (5/5)

Drawbacks:

- 1 TinyOS is becoming dated (2.1.2 was released years ago)
- 2 It has some incompatibility issues from version to version
- 3 Although it is not memory hungry, it however need part of the (small) resource of the mote
- 4 Setting up a complete TinyOS-ready programming environment is not trivial sometimes

Other OSs

Contiki

The Open Source OS for the Internet of Things

RIOT

- 1 **ContikiOS**: a newer OS for WSN and other platforms. It offers support to 6lowpan, RPL, CoAP and a own MAC implementation called *ContikiMAC*
- 2 **RIOT-OS**: generic OS for IoT devices, it help to minimize the hardware-dependent code, it is resource-friendly. It offers real-time capabilities and a form of multi-threading.

Programming WSN motes

Common Approaches

Classical approach: cross-compilation

- 1 **Classical Embedded Systems programming** approach (always valid):
steps:
 - 1 Obtain (or create) a *cross-compiler* toolchain
 - 2 Write the application
 - 3 Use the toolchain to compile the application sources into an executable for the target platform
 - 4 Extract, from the executable, the code and the data creating e.g. a **ihex** or a **srec** file (common file formats which can be read by a programmer)
 - 5 Program the platform using a **programmer** (it can be just a software or a combination of a software and a hardware board)

Study case 1: *MSP430* platforms

- 1 Let's try such approach on a *telosb* platform equipped with a **MSP430** MCU.
- 2 In order to do so, we need the **cross-compilation toolchains** (a ready-to-use version is released by Texas Instruments), the datasheet of the **specific** MCU (**MSP430F1611**) and a board-schematic of the *telosb*

MSP430 (TI) toolchain : demo

- 1 **Objective:** Blinking LEDS
- 2 We need to find which **GPIO** of the MCU is connected to the LEDs on the board (see datasheet and board schematic)
- 3 We will write a simple **Makefile** to speed up the compilation steps
- 4 In this case, for programming the mote we will use the telosb internal **Boot Stage Loader (BSL)** which can be commanded via UART using a python script (**tos-bsl**)
- 5 **LIVE.** Check the code in the folder **MSP430-telosb**

Study case 2: AVR platforms

- 1 Let's change platform. We will now use the IRIS mote, equipped with an **ATMega1281** MCU
- 2 As before, before even start writing code, retrieve all the available material regardin MCU and mote board.

AVR toolchain: demo

- 1 **Objective:** Blinking LEDES
- 2 Find the GPIO port connected to the LEDs
- 3 (optional) Create a Makefile
- 4 Program the node using the mib520 and `avrdude`
- 5 **LIVE.** Check the code in the folder `AVR-iris`

TinyOS programming

- 1 In this section we will use TinyOS to write applications
- 2 We will need to write:
 - A **Configuration** file
 - A **Module**
 - A Makefile

Configuration

Generic Syntax:

configuration *conf-name*

{

uses interface *used-interface-1*;

uses interface *used-interface-2*;

provides interface *provided-interface-1*;

}

implementation{

components *module-name*;

components *used-component-1*;

components *used-component-2*;

...

component.interface-name – > *used-component-1* }

Module

Generic Syntax:

module *module-name*

{

uses interface *used-interface-1*;

uses interface *used-interface-2*;

provides interface *provided-interface-1*;

}

implementation{

command *command prototype* { ...code... }

event *event prototype* { ...code... }

task *task prototype* { ...code... }

other user-define C functions { ...code... }

}

Makefile

A minimal Makefile for a TinyOS application consists only of a component definition and the inclusion of the TinyOS Makefile infrastructure

```
COMPONENT=..  
include $(MAKERULES)
```

A *Blink* example

- 1 Let's create a *Blink* example from scratch (**LIVE**)

A *Blink* example

- 1 Let's create a *Blink* example from scratch (**LIVE**)
- 2 First, create 3 empty files: the **module**, the **configuration** and the **Makefile**

A *Blink* example

- 1 Let's create a *Blink* example from scratch (**LIVE**)
- 2 First, create 3 empty files: the **module**, the **configuration** and the **Makefile**
- 3 Open the **configuration** and fill it with the code required to use the interfaces: **Leds**, **Boot**, **Timer<TMilli>**

A *Blink* example

- 1 Let's create a *Blink* example from scratch (**LIVE**)
- 2 First, create 3 empty files: the **module**, the **configuration** and the **Makefile**
- 3 Open the **configuration** and fill it with the code required to use the interfaces: **Leds**, **Boot**, **Timer<TMilli>**
- 4 Open the **module** and fill it with the application logic

A *Blink* example

- 1 Let's create a *Blink* example from scratch (**LIVE**)
- 2 First, create 3 empty files: the **module**, the **configuration** and the **Makefile**
- 3 Open the **configuration** and fill it with the code required to use the interfaces: **Leds**, **Boot**, **Timer<TMilli>**
- 4 Open the **module** and fill it with the application logic
- 5 Open the **Makefile** and insert the **COMPONENT=** assignment and the inclusion of the TinyOS Makefile

ContikiOS

- 1 Let's now focus on other two modern OS for WSN: *ContikiOS* and *RIOT-OS*
- 2 **LIVE**

WSN Homelab

Description and Assignment

HomeLab: Wireless Sensor Networks

Università degli Studi dell'Aquila
ICT Engineering

Embedded Systems (Dr. L. Pomante)
A.A. 2019/20

Ver. 5.0 22/11/2018

HomeLab: Wireless Sensor Networks

HomeLab WSN Kit:

- 3 sensor nodes Memsic/Crossbow IRIS
- 2 sensor boards Memsic/Crossbow MDA100CB (or BASICSB, or similar)
- 1 programming board Memsic/Crossbow MIB520, USB cable (optional).
- SW to be downloaded and installed: driver MIB520 Windows, VMware Player Windows, virtual machine [DebTos_1](#)
- Official site: www.tinyos.net

Constrained HomeLab: the kit has to be returned in 5-7 days

HomeLab WSN: Environment setup

Installation:

- Follow your “holy book” at the link...
http://tinyos.stanford.edu/tinyos-wiki/index.php/Main_Page
- ...and use one of the methods (basing on your OS and skills) listed in:
http://tinyos.stanford.edu/tinyos-wiki/index.php/Installing_TinyOS

Or

- Follow the method of previous version of HomeLab and download the VM from:
<http://sing.stanford.edu/tinyos/dists/old/xubuntos-2.1-vm.tar.gz>

Or

- Download the new VM (DebTos) from:
https://1drv.ms/u/s!AvY_Lwvi2rN5bE6NIUJ6F8EyD7E
- And use other facilities present in this guide
- **Note: VMWare Workstation Player**
(https://my.vmware.com/en/web/vmware/free#desktop_end_user_computing/vmware_workstation_player/14_0)

7-zip (<http://www.7-zip.org>)

HomeLab WSN: DebTos VM

Why?

- Sometimes you can spend more time setting up the working environment than time to make a real experience with WSNs
- Everything is an exercise, but...
- ... we are trying to made the setup easier as possible, so we have prepared a **Debian 7** (Wheezy) with a working **TinyOS 2.1.2** (“still” latest version up to date), code highlight...

Note

If you use a Windows OS as host machine, you need however to install MIB520 drivers: <http://www.ftdichip.com/Drivers/VCP.htm>

and install it while connecting the MIB520 to an USB port. At the end of the procedure they will be created two virtual COM ports.

HomeLab WSN: DebTos VM



Note:

Username: tinyos, Password: tinyos

HomeLab WSN: Lessons' summary

The following material is a free adaptation of the “Working Group Tutorial” (note: some figures are currently missing in the web pages):

http://tinynos.stanford.edu/tinynos-wiki/index.php/TinyOS_Tutorials

- Lesson 1.1: Getting started with TinyOS
- Lesson 1.2: Modules and the TinyOS Execution Model
- Lesson 1.3: Mote-mote radio communication
- Lesson 1.4: Mote-PC serial communication and SerialForwarder
- Lesson 1.5: Sensing
- Lesson 1.5.1: ADC
- Lesson 1.6 Boot Sequence

- Lesson 1.7 Storage
- Lesson 1.8 Resource Arbitration and Power Management
- Lesson 1.9 Concurrency
- Lesson 1.11 TOSSIM

- Lesson 1.10 Platforms
- Lesson 1.12 Network Protocols
- Lesson 1.13 TinyOS Toolchain
- Lesson 1.14 Building a simple but full-featured application
- Lesson 1.15 The TinyOS printf Library
- Lesson 1.16 Writing Low-Power Applications
- Lesson 1.17 TOSThreads Tutorial
- Lesson 3.2 Rssi Demo

Mandatory

Strongly recommended

Some lessons at your choice

Lesson 1.1: Getting started with TinyOS

([web link](#))

Since we are using a VM with a pre-installed TinyOS environment much of the lesson could be avoided. So:

- physically connect the MIB520 to an USB port (some problems could occur with USB 3.0)
- logically connect the MIB520 to the VM
- by means of a shell go to the folder `/home/tinyos/tinyos-main/apps/Blink`

Since we are using Iris nodes we have:

- `make iris` to compile
- `make iris install mib510,/dev/ttyUSB0` to compile and program the node
- `make iris reinstall mib510,/dev/ttyUSB0` to program the node without compiling
- using the notation `install.ID` or `reinstall.ID` (ID is a 16 bit number without sign so from 0 to 65535) it is possible to assign an ID to each node when programming it

(Please, let you note that we need to use mib510 also if we are actually using a mib520 programming board.)

Lesson 1.1: Getting started with TinyOS

(web link)

Then:

- `make iris`
- `make iris reinstall mib510,/dev/ttyUSB0` (ID is not needed for Blink)

and the node leds should blink like described in the tutorial.

Now, jump to:

http://tinyos.stanford.edu/tinyos-wiki/index.php/Getting_Started_with_TinyOS

and start directly from “4 - Components and Interfaces” to the end of the lesson.

Note

`/dev/ttyUSB0` and `/dev/ttyUSB1` (used later) could appear with a different numbering on your system depending on what USB peripherals you have connected to the VM. In such a case, you can use the `dmesg` command after connected the programming board.

Q: what happened in your Host OS when you plug in the MIB520?

Lesson 1.2: Modules and the TinyOS Exec. Model

(web link)

Follow the whole lesson:

http://tinyos.stanford.edu/tinyos-wiki/index.php/Modules_and_the_TinyOS_Execution_Model

Let's go...

Lesson 1.3: Mote-mote radio communication

(web link)

Follow the whole lesson:

http://tinyos.stanford.edu/tinyos-wiki/index.php/Mote-mote_radio_communication

Let's go...

Lesson 1.4: Mote-PC serial communication and SerialForwarder

(web link)

Follow the lesson:

http://tinyos.stanford.edu/tinyos-wiki/index.php/Mote-PC_serial_communication_and_SerialForwarder

from the beginning to "Sending a packet to the serial port in TinyOS"; have a look at the remaining part of the lesson if you have troubles with Java and see the note in the following slide

Note

The right parameter to be used in this HomeLab is:

- `comm serial@/dev/ttyUSB1:iris` (or 57600 instead of Iris).

Note

Using "&" after shell commands, these will be executed in background avoiding to block the shell.

Q: what happens on /dev/ttyUSB1 or COMx?? Have a look... (Tep 113)

Lesson 1.4: Mote-PC serial communication and SerialForwarder

([web link](#))

Note 11/12/2015: **SOLVED** in ver.4.1

If you get some errors compiling apps that involve java, try the following:

- Open a terminal and type **echo \$CLASSPATH**
 - If you see only:
`:/home/tinyos/tinyos-main/support/sdk/java`
- You have to replace the following line in `/home/tinyos/tinyos-main/tinyos.env` (if you don't see the file, within document explorer, set View->Show Hidden Files)

```
export CLASSPATH=$CLASSPATH:$TOSROOT/support/sdk/java
```

with the following:

```
export CLASSPATH=$CLASSPATH:$TOSROOT/support/sdk/java:$TOSROOT/support/sdk/java/tinyos.jar..
```

- Close all terminals and open again

Lesson 1.5: Sensing

(web link)

Follow the whole lesson:

<http://tinyos.stanford.edu/tinyos-wiki/index.php/Sensing>

Then, to finish this lesson, let you try to modify the proper *DemoSensor* component to change the sensor type (from voltage to light and then to temp).

Note **VERIFY THE PATHS...**

If we compile with *make iris*, we are not specifying any sensor board and so the compiler will use, if needed, the *DemoSensor* component in *opt/tinyos-2.1.0/tos/platforms/micaz* (this a common component for *micaz* and *iris*) that will use the batteries voltage as sensor (the only one available without a sensor board!!!).

In order to use a real sensor board it is needed to specify this, for example, by means of the *SENSORBOARD* environment variable and by compiling in this way (we could also modify the Makefile in the app folder to make such an option the default one):

- If you have a MDA100CB sensor board
 - `SENSORBOARD=mda100 make iris`
- If you have a BASICSB sensor board
 - `SENSORBOARD=basicsb make iris`

Note **VERIFY THE PATHS...**

If you are using a MDA100CB, it is needed another modification to the VM. You have to copy the file *TempImplP.nc* from *opt/tinyos-2.1.0/tos/sensorboards/mda100/cb* to *opt/tinyos-2.1.0/tos/sensorboards/mda100*.